Inducing Succinct Rules in Machine Learning Problems

Ken E. Whelan

A thesis submitted in partial fulfilment of the requirements of the University of Abertay Dundee for the degree of Doctor of Philosophy

June 1998

I certify that this thesis is the true and accurate version of the thesis approved by the examiners

signed [redacted] ..... Date 18 June 1998
studies

Acknowledgements

I wish to thank my supervisors Dr P.E. File, Dr W.B Samson and Mr A.C. Milne for their expert guidance. I would also like to thank other members of the Informatics department for their help: particularly my former supervisor Ms P. Dugard and Dr L.D Natanson. Lastly, I would like to thank Mr E Dolphy and Mr O Coleman serving as examples: particularly in the darker periods

Contents

Abstract

Machine Learning techniques, in particular induction algorithms, have been applied to the field of expert systems development in an effort to overcome the knowledge acquisition bottleneck. Many different induction algorithms have been developed. These utilise a number of different knowledge representations: e.g decision trees and rules. The rule based representation includes systems which utilise both propositional and predicate logic languages. Decision tree and rule based induction employ different knowledge acquisition mechanisms, however both strategies tend to induce complex and inaccurate knowledge from problem domains that contain noise or representational complexity. Previous techniques have concentrated on changing either the knowledge acquisition mechanism or the final ruleset/decision tree to reduce complexity. This thesis presents a new approach that focuses induction on those members of a training set that are likely to provide reliable knowledge. This is achieved by a new measure which is used to identify the most representative examples in a training set. The results of experiments show that this approach produces much simpler rulesets which in some circumstances perform with greater accuracy on unseen data.

Chapter 1

Introduction: Machine Learning and Expert Systems

1.1 Expert Systems

Expert Systems are one of the most widespread of the computer applications to have been developed from the research area of Artificial Intelligence. Artificial Intelligence (AI) itself is a wide research area which is defined by Durkin (1994) as 'A field of study in computer science that pursues the goal of making a computer reason in a manner similar to humans'. It has its roots in research aiming to find an automated reasoning method (Green 1969) and game-playing programs for chess (Shannon 1955). Early AI work also focussed on developing a General Problem Solver (GPS) (Newall et al 1960) which used sophisticated search strategies to solve problems in many different problem domains.

However, the all-encompassing nature of GPS did not take into account the complexity of real world problems, which meant that reaching a solution using this approach could take a very long time. It was soon realised that

some success in solving problems could be achieved by developing systems that were particular to a given problem, provided that specific knowledge of the problem was encoded along with some of the general purpose reasoning strategies used in earlier AI systems. An early example of this system is DENDRAL which was used by NASA to determine the molecular structure of martian soil from mass spectrometer data (Buchanan and Feigenbaum 1978).

It can be argued that a system which contains 'knowledge' about how to solve a particular problem is thus behaving in a similar way to a human expert, thus DENDRAL and its descendants have been termed *Expert systems.* An expert system is defined by Durkin (1994) as 'a computer program designed to model the problem solving ability of experts'. In most expert systems the knowledge of the expert is coded in the form of rules: each rule corresponding to a single item of the expert's knowledge. Generally rules have the following structure:

If
    <these conditions are found to be true>
Then
    <This consequence will become true>

A typical expert system will contain many such rules which are often linked by 'chains' of reasoning i.e. the conditions of one rule are found from the consequences of another rule. These rules form the *knowledge base* of the expert system. The behaviour of the expert system is controlled by the inference engine which governs the chaining structure and determines which rules should be investigated at any given time. Thus an expert system

consists of two main parts: the knowledge base and the inference engine. Researchers found that different knowledge bases could be controlled by the same inference engine because the rules, although containing information about different problems, could be manipulated in the same way. In these systems knowledge about a particular problem can be stored as rules whiles general reasoning strategies can be included in the inference engine.

Many expert systems are in use in a large number of industries including diagnosing medical problems (MYCIN, Shortcliffe 1976), prospecting for minerals (PROSPECTOR, Duda et al 1979) and monitoring gas turbines in power stations (TIGER, Milne et al 1995). The emphasis in expert system development has now shifted to efficient and effective transfer of knowledge from the human expert to the computerised expert system: a process termed *Knowledge Acquisition.*

1.2 Knowledge Acquisition and Machine Learning

1.2.1 Knowledge Acquisition

Knowledge Acquisition for expert systems requires information from many sources: relevant documents, stored data and, of course, the human expert. Indeed most of the knowledge is acquired from the human expert: a process termed *knowledge elicitation.* Knowledge elicitation is one of the most crucial stages in expert systems development, because the resulting expert system will be only as good as the knowledge obtained from the expert. The elicitation process is long and tedious, involving many interviews and case studies

where the knowledge engineer observes the problem solving behaviour of the expert. This most crucial task is thus the most difficult and time-consuming task of expert systems development. Hayes-Roth et al (1983) coined the term 'bottleneck' to describe knowledge acquisition.

'*Knowledge acquisition is a bottleneck in the construction of expert systems. The knowledge engineer's job is to act as a go-between to help build an expert system. Since the knowledge engineer has far less knowledge of the domain than the expert, however, communication problems impede the process of transferring expertise into the program.*'

Since the recognition of the Knowledge Acquisition Bottleneck problem, research has been directed to automating the acquisition process, so that human experts are only required to give examples of their reasoning. Automated knowledge acquisition is designed to determine the problem solving rules from these examples. Machine Leaning techniques have been examined with this task in mind.

1.2.2 Machine Learning

Machine Learning is the generic name given to techniques in Artificial Intelligence which address the problem of automatic knowledge acquisition. Machine Learning can be split into three main research areas :- connectionist machine learning, genetic algorithms and symbolic machine learning. Connectionist machine learning uses techniques that have developed from attempting to simulate neural pathways in the brain (neural networks). Ge-

netic Algorithms are modelled on the principles of Darwinian evolution, and symbolic machine learning techniques are based on human learning and reasoning processes, studied in Cognitive Science. This present study concerns symbolic machine learning, which can be split into four main research areas:

1. Inductive Learning
2. Explanation Based Learning
3. Case Based Reasoning
4. Instance Based Learning

These research areas each have different learning strategies and differences in the type of information to be acquired. There is also a large degree of overlap and hybrid systems have been designed to combine the best features of two different techniques.

Inductive Learning

Inductive Learning is the most researched field of symbolic machine learning. Induction is performed by using a set of heuristics to identify rules from a set of examples. It can also be thought of as a generalisation process: the rules produced have a more general form than the examples used to generate them. Inductive learning is the major focus of this study and is considered in detail in the later chapters.

Explanation Based Learning

Explanation based learning (EBL) (Chi and Kiang 1992, Mooney and Ourston 1989, Wusteman 1992) can be thought of as deductive generalisation. It uses

a single example together with a domain theory that explains how that single example is a member of the concept. The domain knowledge is often represented as a proof tree. Generalisation is achieved by replacing constants in the proof tree by variables and then extracting the values of user defined operational predicates as the concept description. Many EBL systems have been developed using the PROLOG programming language (Wusteman 1992).

For EBL to work it requires a complete domain theory, typically not found in machine learning domains. To overcome the problem of an incomplete domain theory and also the problem of induction ignoring existing domain knowledge, Mooney and Ourston (1989) proposed a hybrid algorithm - Induction Over Unexplained which combines Inductive Learning with EBL. This algorithm reduces the training set being used by a traditional induction system (ID3) (see chapter 3) by removing those attributes explained in the domain theory from the examples constituting the training set. Pazzani and Kibler (1992) use EBL in combination with Inductive Logic Programming (See chapter 4)

Case Based Reasoning

The main difference between case based reasoning (CBR) (Barnden and Srinivas 1992, Kolodner 1992, Slade 1991) and EBL and Induction is that the product of the CBR learning system is not a generalised description consisting of the most prevalent features of a concept, but a series of individual cases representing the experience of the system to date. The learning only occurs by measuring how well a new case is solved by considering the previous cases. If a previous case performs well the new case will not be added to the experi-

ence, but if no previous case performs well the system will attempt to correct it and add the correction to its previous experience. Because CBR has been applied to many temporally driven classification problems, such as planning (Kolodner 1992) the script has often been taken as a knowledge representation, with some capacity for natural language understanding included. The script is a knowledge representation that can model temporal activities by coding each part of the activity in sequence. Barnden and Srinivas (1992) have used CBR and a parallel computer architecture to construct a CBR network. This network allows fast processing of the cases, so that advice on a new case can be given from many past cases and a measure of the relevance of a particular piece of advice can be calculated more quickly.

Instance Based Learning

Instance based learning (Aha et al 1991, Aha 1992) is similar to CBR in that it stores specific instances, not generalised descriptions. However, the representational language consists of attribute value pairs more usually found in induction systems such as ID3 (described in chapter 3). Instances are grouped into categories, and these instances are used to predict the category of a new instance. Categories also contain a measure of how well the instances in that category have done in evaluating new instances. A similarity function is used to compare the new instance with those in the category and this, combined with the performance measurement, gives the correct category for the new instances. The instances used in each category are updated as each new instance is categorised. This update consists of choosing the best instances to be used for each category.

1.3 Aims of the Study

All of the succeeding chapters of this study concentrate on inductive learning. Chapter 2 focuses on the inference mechanisms underlying machine learning and how induction can be perceived as a classification task. Chapters 3 and 4 focus on the two main paradigms of inductive learning. Chapter 5 focuses on the common problems encountered by induction systems, and chapters 6 and 7 focus on a new approach to dealing with some of these problems.

The aims of this work can thus be summarised as follows:

1. Investigate the nature and mechanics of induction algorithms
2. Investigate the behaviour of a well known induction algorithm in a number of different learning situations
3. Describe and explain the reasons for induction of complex rulesets,
4. Develop and test a new method of inducing simple rules.
5. Critically appraise the proposed method.

Chapter 2

Induction and Inference

2.1 Underlying Inference Mechanisms of Machine Learning

The 'Inferential Theory of Learning' described by Michalski (1994) underlies studies pertaining to both symbolic reasoning (Classical Induction, Inductive Logic Programming, Explanation Based Learning, Case Based Reasoning, Abduction) and subsymbolic reasoning (Neural networks, Genetic Algorithms). Michalski views learning as a search in the 'knowledge space' defined by the knowledge representation system used. The search is guided by 'knowledge transmutations' which represent applications of various inference mechanisms. These include inductive generalisation and specialisation, deductive generalisation and specialisation, abduction and prediction, abstraction and concretion, and similization and dissimilization. This study deals mainly with inductive generalisation, inductive specialisation and only briefly with abduction. The reader is referred to Michalski (1994) for a discussion of the other mechanisms.

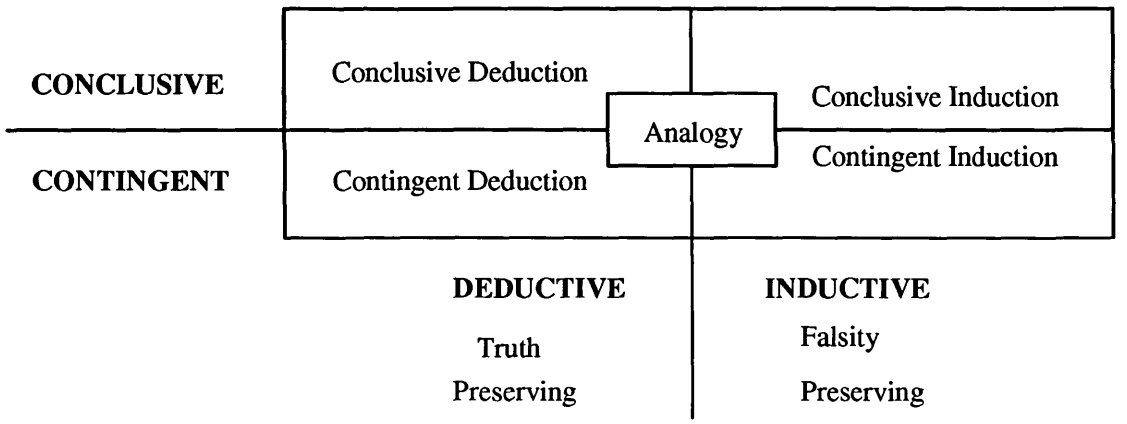

Figure 2.1: fundamental Inference types

2.1.1 Inference Classification

All the inference mechanisms mentioned above can be categorised as illustrated by figure 2.1.

The following equation:

$$P \cup BK \models C$$

represents a 'fundamental equation for inference' where P is the premise (or theory defined as rules etc), BK represents the background knowledge (information already regarded as true by the reasoning system); C represents the consequences: a set of facts produced by processing the information contained in both P and BK. Deductive Inference represents a processing of this equation in a 'forwards' direction, where the desired information is C (the consequences). This operation is truth preserving (see Fig 2.1): if P and BK are both true then C is also true. Inductive Inference is the reverse; deriving premise P from consequences C and background knowledge BK.

Induction is falsity preserving: if C is not true then P cannot be true. Figure 2.1 also divides inference into Conclusive and Contingent categories. This distinction is based on the strength of the Premise or Conclusions obtained by the inference process. Conclusive inference produces true Conclusions from true premises, contingent inference weakens the truth value of the inference process. Conclusions and premises may be true in some situations but not in others. Traditional Induction algorithms (see chapters 3 and 4) perform conclusive inductive inference.

2.1.2 Inductive generalisation and specialisation

Each attribute or relation described in the training set of examples or in the background knowledge has a 'reference set' of those examples for which the attribute or relation is true, i.e. the attribute or relation appears in the example. This condition is related to the concept of 'coverage'. Coverage refers to the number of positive and negative examples which are true for any concept description. Positive examples belong to the same class as the concept description, negative examples belong to any of the other classes found in the problem. Conclusive Induction Algorithms attempt to achieve complete coverage of positive examples and no coverage of negative examples. Contingent Induction relaxes this condition to achieving coverage of a large percentage of positive examples and a small percentage of negative examples. Inductive generalisation is a knowledge transmutation which extends the coverage of a current concept description (this extension may apply to negative examples as well as positive examples). Inductive specialisation reduces the coverage of a current concept description (again for both sets of examples).

2.1.3 Abduction (or Abductive Derivation)

Abduction is described by Michalski (1994) as 'a process of creating the 'best' explanation of (a) given fact(s)'. Abduction often traces back chains of implicative statements, having many antecedent facts. In this way it treats the modus ponens deductive rule backwards by inducing A from A $\rightarrow$ B and B. For example:

from the rule

$\forall$ X drunk(X) $\rightarrow$ stumbling(X)

and witnessing a stumbling person, i.e. stumbling(X) is true, it is abduced that this person is drunk.

A single fact may give many possible abductive derivations e.g wounded(X) or sunstruck(X) may be abduced if the rules:

$\forall$ X[sunstruck(X) $\rightarrow$ stumbling(X)] and
$\forall$ X [wounded(X) $\rightarrow$ stumbling(X)]

are also contained in the background knowledge for the example above. Abduction may thus extend the number of facts which could be used in concept descriptions. Abduction can be used in combination with induction (see section on Inductive Logic Programming), where user interaction can be used to verify the derived facts.

2.2 Inductive Learning as a Classification Task

The induction of more general concept descriptions can be viewed as a classification task (Brieman et al 1984) in which a training set of examples is

analysed to produce a more general description of the classifications described by this training set. Each example is itself a single instance of this classification problem: where a conjunction of *vectors* describing important features of the problem are assigned a particular value of a vector which is used to classify the example. The resulting *classifier* will be a collection of descriptions each containing a subset of the feature vectors used to describe the examples. Each description will group training examples into subgroups: with each member example containing all of the vectors present in the relevant classifying description. A description is said to *cover* a training example if all of the vectors in the description are also found in the example. The goal of this classification task is to find as small a set of descriptions as possible which correctly group the training set according to the classification of the examples. The search for a good single description is to find a collection of as few vectors as possible which correctly describe as many training examples as possible.

This grouping process can be visualised by plotting the examples as a graph on which the axes are the values of the vectors used to describe the features of an example. Figure 2.2 shows a graphical representation of the training set illustrated in table 2.1. The classification task in this illustration is to identify 'large' triangles (where large is an area > 50)

The correct grouping for this problem would be found by the following description:

large triangle = true if shape = triangle AND area > 50

This illustration is very simple: most classification problems contain more

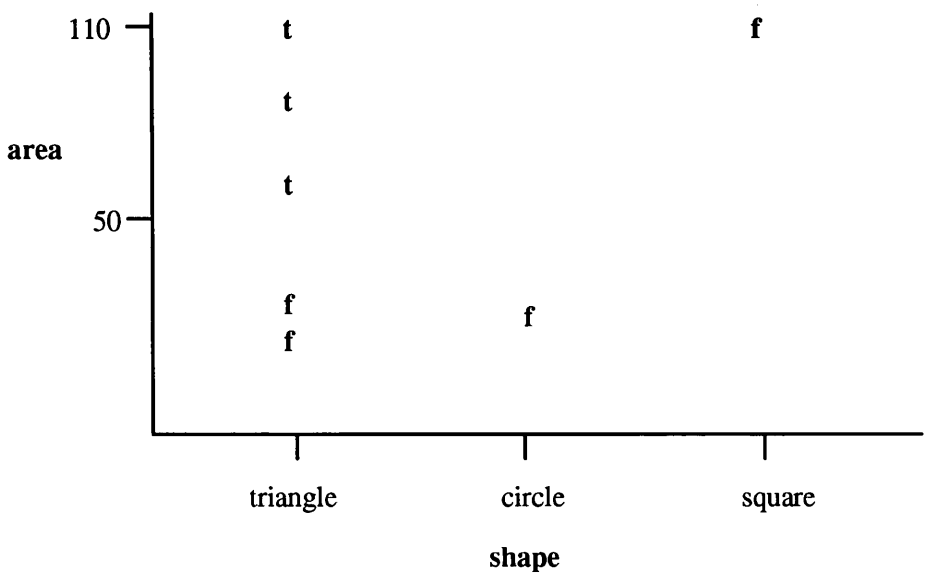

Figure 2.2: A Graphical Representation of a shapes training set

class	Vector	
	shape	area > 50
true	triangle	56 (y)
false	circle	34 (n)
false	square	103 (y)
true	triangle	109 (y)
true	triangle	76 (y)
false	triangle	32 (n)
false	triangle	21 (n)

Table 2.1: A simple shape classification problem

than two feature vectors and have classification vectors with more than two values. Graphical illustrations of problems with more than three vectors is difficult, requiring a new dimensional axis for each vector, with the examples occupying an N-dimensional space. However this example does illustrate that the values of feature vectors can be of different types; nominal vectors containing non-ordered categories (e,g, shape) and ordered vectors containing values which represent a succession (e.g the area of the shape). Ordered vectors can be discrete (e.g. integers) or continuously valued (e.g. real numbers). Classification vectors can also have nominal or ordered values. In this case the description used to classify the shapes problem is a rule. Classifiers can use a number of different formalisms to describe their representations. These formalisms are used to categorize the Inductive Learning paradigms discussed in this study. Two main formalisms are used: decision trees and decision rules (also known as decision lists). The next two chapters of this study describe these different paradigms, with a further categorisation of decision rule induction based on the knowledge representation used to describe both the rules produced and the training examples used. The issue of different vector types is also discussed in these further chapters where the knowledge representation used by each category is discussed in more detail.

Chapter 3

Decision Tree Induction

3.1 Introduction

All of the induction algorithms in this chapter conform to the learning paradigm known as Top Down Induction of Decision Trees (TDIDT). A decision tree classifies objects by 'pathways' that connect nodes representing questions concerned with the feature vectors of a learning problem. Decision trees are hierarchical and represent increasingly more specialised splits of the problem population. The most general split is found at the 'root' of the tree. The vector forming this node has its values in the form of branches which lead to more specialised nodes. Each new node has further branches giving the tree a recursive form. Each branch of the tree is finally terminated by a leaf node which consists of the relevant value of the classification vector. and represents a pathway from the root node to a classification value. A simple decision tree for the shapes problem described in the previous chapter is given in figure 3.1. The knowledge representation used by TDIDT algorithms terms each vector as an 'attribute' with the examples in the training set de-

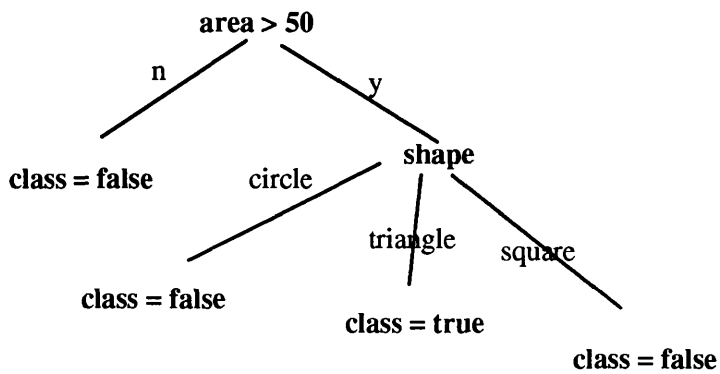

Figure 3.1: A decision tree for classifying 'large' triangles

scribed using attribute value pairs, where a single value of the classification attribute is connected to a conjunction of attribute value pairs describing important features.

Quinlan (1993) gives a general description of decision tree induction. To construct a decision tree from a set T of training cases, let the classes be denoted $C_1, C_2, \ldots, C_j$. There are three possibilities:

1. T contains one or more cases, all belonging to a single class C_j:

   The decision tree for T is a leaf identifying class C_j.

2. T contains no cases:

   The decision tree is again a leaf, but the class to be associated with the leaf must be determined from information other than T. For example the leaf might be chosen in accordance with some background knowledge of the domain, such as the overall majority class.

3. T contains cases that belong to a mixture of classes:

   In this situation, the idea is to refine T into subsets of cases that are, or

seem to be heading towards, single class collections of cases. A test is chosen based on a single attribute, that has two or more mutually exclusive outcomes $O_1, O_2, \ldots, O_n$. T is partitioned into subsets $T_1, T_2, \ldots, T_n$, where T_i contains all the cases in T that have outcome O_i of the chosen test. The decision tree for T consists of a decision node identifying the test and one branch for each possible outcome. The same tree building machinery is applied recursively to each subset of training cases, so that the ith branch leads to the decision tree constructed from the subset T_i of training cases.

Decision tree algorithms generally construct decision trees which classify the training sets used to describe the learning problem with 100% accuracy. A splitting criterion determines which attribute should be placed in a particular node in the tree. The attribute chosen is that which minimises the 'impurity' of the training set. Impurity is defined as the number of misclassifications generated by testing classification by processing a single attribute (Michie et al, 1994). Measures of impurity are generally numerical, for example the information based measure used in ID3. Decision tree induction conforms to the inductive specialisation inference mechanism where the independent values of the first attribute placed in the decision tree will cover more training examples than subsequent attribute values down each branch. Reduction of impurity and inductive specialisation are used to (hopefully) produce the shortest branch which will correctly classify the class attribute value placed at the end of it. The concept of coverage again underlies this process i.e. the branch should not cover any negative examples. The full

set of branches which lead to a particular class should also cover all training examples belonging to that class.

3.2 ID3

ID3 (Quinlan 1986) is the most commonly used decision tree induction algorithm. In fact, all of the other systems described in this chapter are only updates and modifications to the process described here. As mentioned above, ID3 uses an information theoretic approach to decision tree induction. Quinlan (1993) defines this approach as measuring the information gained by choosing an attribute to split the training set. The heuristic used by ID3 considers the information gained from each attribute in turn, and selects the attribute which conveys the highest information gain for splitting the set. The information contained within the initial training set S is the average of the information required to classify a single training example belonging to this set; and is given by the following equation.

$$info(S) = -\sum_{j=1}^{k} \frac{freq(C_j, S)}{|S|} \times log_2(\frac{freq(C_j, S)}{|S|}) bits$$

where *freq(C_j,S)* refers to the number of examples in set S that belong to class C_j and $|S|$ refers to the total number of examples in the training set. The best attribute for splitting the set is determined by substituting each attribute for test X in the following equation:

$$info_X(S) = \sum_{i=1}^{n} \frac{|S_i|}{|S|} \times info(S_i)$$

where S_i is a subset of S according to the ith value of the attribute being used as test X. The gain found from placing this attribute at the appropriate node in the decision tree is:

$$gain(X) = info(T) - info_X(T)$$

3.2.1 An illustration of Decision Tree Induction

The shapes example outlined in Chapter 2 will be used here to illustrate how the decision tree in figure 3.1 is induced from the examples in table 2.1. The first step is to calculate the information contained within the initial training set. There are seven training examples and two classes in this problem: 3 belonging to class *true* and 4 belonging to class *false*. The information entropy calculation is thus:

$$\text{info(S)} = -3/7 \times \log_2(3/7) - 4/7 \times \log_2(4/7) = 0.985$$

The next step is to calculate the information gain for each attribute so that the attribute which gives the highest information gain can be placed as the first node of the decision tree. In this case the calculations are for *shape* and *area* > *50*. These calculations are shown below:

$$\begin{aligned}
\text{info(shape)} &= 5/7 \times (-3/5 \times \log_2(3/5)) - 2/5 \times \log_2(2/5)) \\
&+ 1/7 \times (-0/1 \times \log_2(0/1)) - 1/1 \times \log_2(1/1)) \\
&+ 1/7 \times (-0/1 \times \log_2(0/1)) - 1/1 \times \log_2(1/1)) \\
&= 0.694 \\
\text{info(area} > 50) &= 4/7 \times (-3/4 \times \log_2(3/4)) - 1/4 \times \log_2(1/4)) \\
&+ 3/7 \times (-0/3 \times \log_2(0/3)) - 3/3 \times \log_2(3/3)) \\
&= 0.463
\end{aligned}$$

The information gain for each attribute is calculated:

gain(shape) = 0.985 - 0.694 = 0.291
gain(area > 50) = 0.985 - 0.463 = 0.522

Thus the attribute *area > 50* would be used to split the training set into the subsets shown in tables 3.1 and 3.2. The attribute area > 50 is then placed at the root of the tree and a branch for each value is added. This splitting procedure is repeated in a recursive manner for each subset, so that further nodes, or indeed class attributes can be added. In this case the subset for area > 50 = no contains only examples belonging to the class false. Thus the class attribute (class = false) is added to this branch thus terminating it. Repeating the attribute selection procedure for the area > 50 = yes subset results in the attribute shape being added as the next node for the triangle branch. (this is a trivial case since the shape attribute is the only remaining attribute). Splitting the subset represented by area > 50 = no (given in table 3.1) by the shape attribute terminates the tree growing process resulting in the tree illustrated in figure 3.1.

In the above example all the attributes have nominal values. However, ID3 can also use continuously valued attributes to split training sets. This is achieved by finding the best binary split of attribute A i.e A > V or A =< V. This value V is found by applying the information gain heuristic to the midpoints of the ranges between consecutive values of A found in the training set. To achieve this the values of attribute A are sorted in ascending order and the midpoint between any two consecutive values is found by summing them and then dividing by two:

class	Vector	
	shape	area > 50
true	triangle	56 (y)
true	triangle	109 (y)
true	triangle	76 (y)
false	square	103 (y)

Table 3.1: Training examples conforming to the 'area > 50 = yes' subset

class	Vector	
	shape	area > 50
false	triangle	32 (n)
false	triangle	21 (n)
false	circle	34 (n)

Table 3.2: Training examples conforming to the 'area > 50 = no' subset

midpoint	info(area)
26.5	0.128
33	0.291
45	0.521
66	0.148
89.5	0.006
106	0.198

Table 3.3: Information Gain Calculations for the "area" attribute

$$\frac{v_i + v_{i+1}}{2}$$

Table 3.3 gives the information gain calculated for all the midpoints of the area attribute. Thus this procedure would identify the value 45 as the best split for the attribute *area*. The node representing the area attribute would have two branches: one representing the $area =< 45$ choice and the other representing the $area > 45$ choice.

3.3 C4.5

C4.5 (Quinlan 1993) is an information based decision tree algorithm which is a modern descendant of ID3. It uses the coverage principle in a unique way. First a subset of training examples is randomly chosen to represent each classification set. The impurity measurement is applied to the subset

only and a decision tree is produced which covers only these subsets. The decision tree is then tested on those examples not included in the initial classification. If the tree also covers these examples no further processing is required. If misclassified examples are found, then these are added to the initial subsets and a new tree is derived. This process stops when the resulting tree completely covers the training set.

Decision tree induction can be severely impaired by training sets which are incomplete: i.e. there are examples which contain unknown attribute values. If an decision tree induction algorithm cannot construct trees from examples containing unknown attributes a large part of the training set may convey no information to the splitting process, thus a large number of unseen cases may be unclassifiable (Quinlan 1993). To overcome this problem C4.5 contains techniques for inducing from incomplete information. This is particularly relevant for many real world problems because missing information is common due to poor data collection or an inability to perform certain tests needed to determine certain attribute values. Quinlan (1993) adopts a probablistic approach where the information heuristic used to determine the suitability of a particular attribute is altered to take into account that fraction of the training examples containing an unknown value for this attribute. The gain measurement is altered to:

$$\text{gain(X) is F} \times (\text{info(T)} - \text{info}_X\text{(T)})$$

where T is the training set and F is the fraction:

$$\frac{\textit{cases with value of } X \textit{ known}}{|T|}$$

and $|T|$ is the total number of examples in the training set.

The information heuristic used to identify the best attribute to split the training set ($\text{info}_X(\text{T})$) is amended so that the cases with unknown values of X are treated as a separate group. When attribute X has been determined, and T is found to contain cases with unknown values of X, the subsets T_i are amended so that classification can be achieved using probabilities. To achieve this each case in a subset is given a probability weight. If X for that case is known then this weight = 1: each unknown case is added to every subset with a weight which is calculated by the fraction:

$$\frac{number\ of\ known\ cases\ in\ T_i}{|T| - number\ of\ unknown\ cases}$$

This subset is now considered to contain a fractional number of cases: so that $|T|$ for any future information calculations is a sum of the probability weights of all the cases in this subset. This procedure is generalised so that a fractional case from a previous split is assigned to a further subset by multiplying the probability weight from the current split by the sum of the weights of cases in T known to have the relevant value of X divided by the sum of weights of cases with any known value of X. If an unseen example to be classified by the decision tree has unknown values for an attribute, all possible paths are followed, eventually giving a probability distribution for the predicted classification. C4.5 chooses the class with the highest probability as that used to classify the unseen example.

C4.5 also contains a procedure to reduce the number of branches if a categorical attribute with many possible values is chosen as the best attribute.

The values are grouped into subsets and the information gain heuristic is applied to training examples corresponding to these subsets rather than a single value of the attribute.

3.4 CART

CART (Classification and Regression Trees) (Brieman et al 1984) is a decision tree induction algorithm which splits the training set into two subsets at each node. This binary tree is a special case of the multiple-branch tree produced by ID3 and C4.5. At each node the test induced by CART must conform to the following standard set of 'questions':

1. Each split depends on the value of only a single attribute
2. If an attribute is ordered, then the question has the form:

   {Is A $\leq$ V?} e.g. Is area $\leq$ 45 ?

3. If an attribute is categorical, with possible values $\{v_1,v_2,...,v_n\}$, then the question has the form:

   {Is A $\in$ S?} where S is a subset of $\{v_1,v_2,...,v_n\}$
   e.g. Is shape in {circle,square}

The two branches from each node, representing the Left and Right subtrees respectively, are labelled yes and no: yes, for the Left subtree and no,

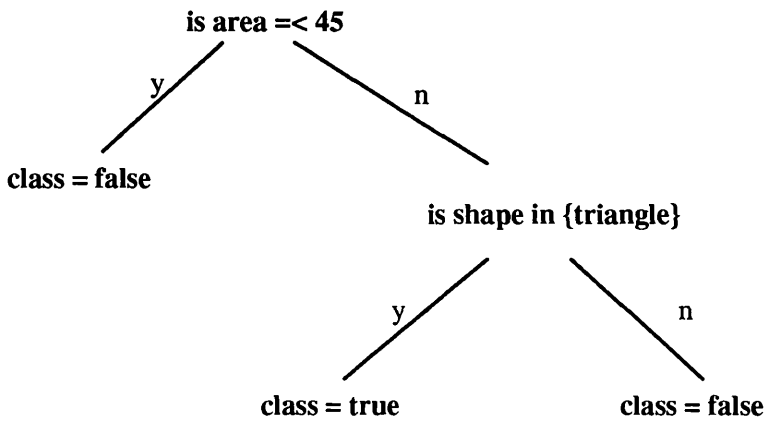

Figure 3.2: A binary tree for classifying 'large' triangles

for the Right subtree. The midpoint used to split ordered attributes is found by applying the same procedure as discussed for ID3. CART might have induced the binary tree illustrated by figure 3.2 for the shapes training set used above.

As a measure of impurity, for deciding on the best attribute to split the training set, CART uses the GINI measure:

$$Gini(c) = 1 - \sum_j p_j^2$$

where c is the current node and P_j is the probability of class j in the set to be split at node c (initially the whole training set when the root node is being determined). This measure is also used to calculate the attribute and value(s) to be used as the question forming node c. The impurity of categorical attributes uses the attribute subset method, which is also used by C4.5 for categorical attributes containing many values. This method calculates the impurity measure for each possible subset of values and picks tthe best one.

CART picks the best attribute and value description once all the attribute splits or subsets have been evaluated by the GINI measure. As for ID3 and C4.5 the subset splitting is recursive. The branch termination criterion for CART uses a significance measure: a set is not split if impurity cannot be significantly reduced by any split. In the case of the subset represented by a terminal node containing examples belonging to two or more classes, the class value of the most common class is taken as the class assignment for that terminal node. If the maximum number is shared by two or more class values then an arbitrary choice from these class values is used as the class label.

Decision trees are considered to be difficult for humans to understand, especially if they are very large and contain many sub-branches emanating from each node (Cendrowska 1987). To overcome this many induction systems present derived concept descriptions as rules. Indeed C4.5 includes a procedure for displaying each branch as a single rule. This procedure also allows a different form of 'pruning' where attribute value pairs may be dropped from each rule. C4.5 and CART both implement various 'pruning' techniques to improve the performance of the induced decision tree. These and other techniques used to 'fine tune' decision trees and rule sets are discussed in chapter 5. The next chapter discusses those algorithms which present their answers as rules rather than decision trees.

Chapter 4

Induction of Decision Rules by Generalisation and Specialisation

4.1 Introduction

The induction algorithms described in this chapter are members of the "generalisation and specialisation" family of induction algorithms which represent solutions to the "general covering problem" (Hong 1985), a more formal description of the search for rules among the training examples. The covering problem is a search for a cover, *cover(e/E)*, which takes the form of a rule that explains how the example e is different from the set of examples E. Finding such a cover is a complicated problem which is described as NP-Hard. Thus a search for an optimal solution will not be be possible. The solutions presented by the algorithms described in this paper are "good" solutions which employ heuristics to focus the search on those knowledge items which offer better explanations of the examples. The search space for this problem

includes every possible combination of the knowledge items which comprise the examples: the search strategies employed control the generation of these combinations as well as evaluation of how good the combinations generated are at distinguishing and describing the examples. The two operations used to generate combinations of knowledge items are termed generalisation and specialisation. Generalisation produces combinations which can be used distinguish and describe more examples: specialisation produces combinations which can be used to distinguish and describe fewer examples. Each induction algorithm represents an attempt to combine these operations to produce rules as efficiently as possible. The evaluation process allows the generalisation and specialisation operations to be directed at the most promising combinations. This chapter discusses each algorithm in terms of the search strategy used.

4.2 Induction of Decision Rules using Propositional Logic

4.2.1 The VL_1 Representation Language

The VL_1 language is used to describe examples of the solutions an expert gives when presented with various problems in a learning domain. VL_1 is a *multi-valued variable logic language* (Michalski and Chilausky 1980) which is analogous to the attribute-value pair knowledge representations found in the ID3 literature. In this language the term *variable* is used to describe the knowledge items termed attributes in the ID3 literature. Variables are multi-valued because each one has a set of values associated with it. Exam-

ples are presented as conjunctions of variables with each variable linked to one value from its value set. Examples are always assigned to a *class* which represents the decision reached by the expert if the conditions represented by the conjunction of variables was found. Examples are often presented in tables: table 4.1 shows the variable values and classifications for a small number of mushrooms. Conjunctions of variables in concept descriptions can contain a list of values linked by logical OR. The basic knowledge item in VL_1 is the *selector* which is defined as follows:

$$[x_i \ \# \ R]$$

where x_i is a variable, R is the *reference* (a list of one or more variables from the value set of x_i), and "#" denotes a relational operator from the set {=,<>,>=,=<,<,>}. A conjunction of selectors is termed a *complex* and a disjunction of complexes is termed a *cover*. The goal of the AQ series of induction algorithms to find a set of covers which succeed in distinguishing different classes of examples. The covers are linked with a classification by using the "::>" operator. Covers presented in this way are used as concept recognition rules for classifying unseen examples. The concept recognition rule for classifying Fly Agaric mushrooms is given below:

[cap_colour = red] & [spots = yes] & [spot_colour = white]
∨
[cap_colour = orange] ::> [class = fly agaric]

All of the algorithms described in this section use the VL_1 representation

cap_colour	cap_diameter	spots	spot_colour	stem_colour	ring	volva	class
red	15	yes	white	white	yes	yes	fly agaric
orange	12	no	none	white	yes	yes	fly agaric
olive	15	yes	white	olive	yes	yes	death cap
olive	12	yes	white	white	yes	yes	death cap
red	10	yes	yellow	yellow	yes	yes	ceasar's
red	8	no	none	white	no	no	sickener

Table 4.1: VL_1 expressions for describing six mushrooms

language. This is in contrast to the algorithms described in the next section which use a richer representation language which can include predicates and functions as well as attributes. The following algorithms all search for covers within the description space built from the examples. A discussion of the different approaches used by each algorithm is presented below, along with a discussion of the problem(s) that the algorithm was designed to overcome.

4.2.2 AQ11

The AQ11 (Michalski and Chilausky 1980, Mickalski and Larson 1978) algorithm is the simplest version of the generalisation-specialisation solution to the general coverage problem. The steps presented below are the basic steps used by the first two algorithms in this section while HCV uses the extension matrix approach. This algorithm is also referred to as AQR (Clark and Niblett 1988)

Let POS be a set of positive examples of class C
Let NEG be a set of negative examples of class C

Procedure AQR(POS, NEG)
 Let COVER be the empty cover
 While COVER does not cover all examples in POS
 Select a SEED (A positive example not covered by COVER).
 Let STAR be STAR(SEED, NEG) (A set of complexes that cover SEED but cover no complexes in NEG)
 Let BEST be the best complex in STAR according to user-defined criteria
 Add BEST as an extra disjunct to COVER
 Return COVER.

Procedure STAR(SEED, NEG)
 Let STAR be the set containing the empty complex
 While any complex in STAR covers some negative examples in NEG,
 Select a negative example E_{neg} covered by a complex in STAR.
 Specialize complexes in STAR to exclude E_{neg} by:
 Let EXTENSION be all selectors that cover SEED but not E_{neg}
 Let STAR be the set $\{x \wedge y | x \in STAR, y \in EXTENSION\}$
 (construct a new star by specializing the complexes found in the old star by those complexes which do not cover the negative example)
 Remove all redundant complexes in STAR
 Repeat until the size of STAR $=<$ *maxstar* (A user defined maximum)
 Remove the worst complex from STAR
Return STAR.

This algorithm acquires covers for each class in a multi-concept learning system in turn. The set NEG will contain all examples which belong to classes other than the class being acquired at the time. AQ11 implements this AQR algorithm by retaining those complexes which are maximally general (usually those complexes which contain the fewest number of selectors) and those complexes which are maximally specific (usually those complexes

containing the largest number of selectors): all other complexes can be generated from these two sets. Thus the number of complexes is kept to a minimum. In this way the combinatorial nature of the description space is overcome. Examples are presented to the complexes in these two sets in an attempt to specialise the maximally general set and generalise the maximally specific set. Examples are presented according to a distance measure from the seed positive example, measured by counting the number of attributes with different values. If the chosen example is positive the complexes in the maximally specific set are generalised to include coverage of the new positive example. If the chosen example is negative, the complexes in the maximally general set are specialised to exclude coverage of the new negative example. Specialisation involves adding those selectors in the new positive example to an existing complex to extend its coverage to that positive example. Generalisation involves removing selectors which allow any complex to cover the negative example. After each knowledge transformation step the sets are updated to contain only the maximally specific and maximally general complexes. The best cover is found by disjunctively combining the complexes when these two sets become the same (i.e when the generalisation and specialisation processes meet)

AQ11 also applies a confirmation measure to selectors in the examples and rules. This measure is used to manage uncertainty where selectors with a lower *weight* (measure of importance) can be discounted if there values are unknown.

4.2.3 HCV - An Extension Matrix Induction Algorithm

The HCV induction algorithm (Wu 1993) uses the Extension Matrix solution to the general coverage problem to induce decision rules from training examples described in the same variable-valued logic language as the AQ series of algorithms discussed above. The extension matrix approach extracts rules by searching for disjunctive expressions that contain attribute-value pairs which distinguish examples in the negative subset from examples in the positive subset. An extension matrix is produced for each positive example in turn. This extension matrix contains all the attribute value pairs which distinguish the positive examples from all the negative examples. Disjunctive expressions are found by combining these attribute value pairs by logical OR. The induction of rules by the extension matrix approach can be explained by the following example (Zeleznikow et al 1995)

Tables 4.3 and 4.4 represent the negative examples and positive examples, respectively, of a small training set taken from the legal domain. The columns of each table represent the attributes used to define the domain, and each row represents a single training example defined by its appropriate attribute-value pairs. The attributes used to define the domain are: Age, Health and Parenting Desire; and the classifications ascribed to each example are *can work* (positive) and *cannot work* (negative).

The extension matrix produced for example f1 (EMf1) is shown in table 4.5. Initially the extension will contain all attribute values which are found in the negative examples. By comparing these values with the attribute values

Example ID	Age	Health	Parenting Desire
n1	young	minor perm problem	full time
n2	young	minor perm problem	full time
n3	middle age	major perm problem	none
n4	elderly	minor perm problem	none

Table 4.2: Negative examples - Cannot Work

Example ID	Age	Health	Parenting Desire
f1	young	excellent	none
f2	middle age	excellent	none

Table 4.3: Positive examples - Can Work

found in example f1, the attribute values which can be used to distinguish f1 from all the negative examples are found: the shared attribute value pairs are declared "dead" and take no further part in rule extraction.

A disjunctive expression extracted from this extension matrix will produce a rule for testing whether unseen examples belong to the negative class (*cannot work*) in this case. A disjunctive expression is built by examining "paths" through the extension matrix. A path consists of one attribute value pair from each row of the extension matrix. A single extension matrix can thus contain many paths. Two are given below:

path1: [Age = middle age, Age = elderly, Parenting Desire = full time, Parenting Desire = full time]

Age	Health	Parenting Desire
dead	minor perm problem	full time
dead	major perm problem	full time
middle age	major perm problem	*dead*
elderly	minor perm problem	*dead*

Table 4.4: Extension Matrix for example f1

path2: [Health = minor perm problem, Health = major perm problem, Health = major perm problem, Health = minor perm problem]

The rules produced from these paths are:

if [Age = middle age or elderly] OR [Parenting Desire = full time] then [class = cannot work]

if [Health = major perm problem or minor perm problem] then [class = cannot work]

Such rules are found by creating extension matrices for each example in the positive subset. The term *intersecting group* is used to denote a set of examples whose extension matrices share a common path. A rule extracted from this common path will distinguish all the positive examples in the intersecting group from all the negative examples.

The goal of the HCV algorithm is to find the most compact rule from each intersecting group. Extension matrices are produced for each example so that the positive subset can be partitioned into these intersecting groups. Rules

can be extracted from the common paths found in the extension matrices of each intersecting group. An intersecting group may contain more than one common path; the most compact rule can be found by choosing the common path which contains the fewest attributes. The rule which can be used to distinguish all of the positive examples from all of the negative examples is the conjunction of each rule produced from the intersecting groups.

The extension matrix approach, incorporated in the HCV algorithm, represents a simple inductive learning system which is computationally inexpensive and produces rules which are consistently more compact than the trees produced by algorithms of the ID3 type (Wu 1993).

4.3 Induction of Decision Rules using Predicate Logic

All of the above induction algorithms use a propositional logic language based on attribute-value pairs to represent the examples which are to be searched for decision rules and also to represent the decision rules themselves. This representation is sufficient to describe many domains of expert reasoning. However, induction in an increasing number of expert domains requires a language which is capable of describing relationships between different components of an example. This has led to the development of a number of induction systems which represent examples in a First Order Predicate Logic language. The language used and the algorithms are discussed below.

4.3.1 The VL_{21} Representation Language

The VL_{21} language (Michalski (1980)) is a multi-valued first order predicate logic language which represents examples as *structured objects* which consist of a number of components. These components are uniquely identified by *variables* (This is in contrast to VL_1 where a "variable" is analogous to an attribute in the ID3 literature). The basic knowledge item in VL_{21} is still termed a selector; however the selectors in VL_{21} are more complex than selectors found in VL_1. A VL_{21} version of the first fly agaric mushroom used to illustrate the VL_1 language is given below:

```
exists  Mushroom,Stem,Cap,Spots [has(Mushroom,Cap,Stem,Spots)]
        [shape(Cap) = circle][diameter(Cap) = 15]
        [colour(Cap) = red][ontop(Cap,Stem)]
        [length(Stem) = 10][colour(Stem) = white]
        [has_spots(Cap,Spots)][colour(Spots) = white]
        [has(Stem,Ring)][has(Stem,Volva)] ::>
        [class(Mushroom) = fly agaric]
```

This mushroom has five components, identified by the variables Cap, Stem, Spots, Ring and Volva. Variables always form the arguments of selectors. Selectors can be simple or complex; simple selectors are attribute value pairs which describe one property of one sub-object of the example. Complex selectors describe relationships between different sub-objects of an example. Selectors can be of three types :- attributes (simple selectors), predicates or functions (complex selectors). These types are described in table 4.6.

The examples above show the differences between the forms of the three

example	selector type	English translation
[diameter(Cap) = 10]	attribute	the diameter of the Cap is 10 cm
[ontop(Cap,Stem)]	predicate	the Cap is above the Stem
[distance(Part1,Part2) = 4]	function	the distance between Part1 and Part2 is 4 cm

Table 4.5: Types of selectors possible in VL_{21}

kinds of selectors. Attributes can only take one variable as an argument and have one value. Predicates can have any number of variables as arguments and always have the value true. Functions are similar to predicates in having many variables as arguments but these are related to an actual value (other than true or false). The value of an attribute or function can include internal disjunction which takes the form of a list where the individual values are linked by OR. The referee of an attribute or function can be linked to its value by the same set of relational operators as the selectors in VL_1

Selectors are combined to form expressions by both disjunction and conjunction. An expression represents the full collection of properties that comprise a training example or concept description. There are two types of expression: atomic expressions (a_expressions) and compound expressions (c_expressions). A_expressions are used to represent training examples and cannot contain disjunctive statements (which describe possible choices of attribute values) Compound expressions are used to represent concept rules

and so can contain disjunction. The concept rule for recognising fly agaric mushrooms is given below:

[colour(Cap) = red][has_spots(Cap,Spots)] [colour(Spots) = white]
∨
[colour(Cap) = orange] ::>
[class(Mushroom) = fly agaric]

Testing expressions for coverage presents a problem which is unique to Predicate Logic induction systems, the problem of pattern matching. Two selectors in VL_{21} are said to match if the following conditions are met (Esposito et al 1992):

- The referee of the selectors must be the same
- A consistent binding must exist between the variables appearing as arguments in the selectors.
- The reference set of the selector to be matched must be less general than the selector it is to be compared with.

Matching expressions consisting of more than one selector is more complicated described as NP-Complete by Esposito et al (1992). This match is achieved by splitting the expression into two "sub-expressions". The first sub-expression contains selectors ordered as follows: the first selector is the selector with the largest number of arguments; the next is the selector with next largest number of arguments, with the extra condition that none of the variables in this selector have appeared in the any previous selectors. The other sub-expression contains all the remaining selectors. This allows matching to concentrate on less-likely selectors so that the matching process can

give up quickly if two expressions do not match .(Selectors containing many arguments are rarer than selectors containing few arguments.)

4.3.2 INDUCE

The INDUCE algorithm (Michalski 1983) induces concept recognition rules from training examples presented as VL_{21} a_expressions. The algorithm searches for concept recognition rules in the space of possible expressions which can be generated from a single positive example. The same principle of coverage is used to guide the search for useful expressions. The most useful expression is one which covers all the positive examples and none of the negative examples. Initially all selectors in the seed example are processed for coverage individually and a user-defined percentage of "best"selectors is retained for future coverage evaluation. Each selector in this list of "best" selectors (termed a partial star) is specialised by the addition of one selector from the original set. Specialisation is used to limit coverage of the negative examples, and the result of this process is the set of shortest a_expressions which cover no negative examples. Specialisation of this nature is combinatorial but the retention of only a percentage of the best expressions in the partial star prevents generation of a large number of new expressions. The application of another user-defined criterion terminates the specialisation process when a specified number of consistent expression are found. The expressions produced by this step are generalised by a powerful set of generalisation rules that either drop conditions from the expressions or add internal disjunction to the values of attributes and functions. These rules are guided by background knowledge that describes the type and possible

values of the selectors used to define the problem space, allowing each generalisation rule to operate on the correct type of selectors. Generalisation is used to extend the coverage of expressions in the partial star so that as many positive examples are covered as possible. The resulting concept rules are disjunctions of the expressions that remain after the generalisation step. These concept rules may not cover all of the positive examples. In this case, all the examples covered by the existing concept rules are removed, a new seed is chosen at random from the remaining examples and the algorithmic steps are repeated for this new seed. The resulting concept ruleset will thus cover all positive examples. The algorithm used at each learning iteration is also known as the STAR algorithm, because the resulting concept rules are defined as the star(e|E) i.e. all concept descriptions from the seed example e which do not cover any examples in set E (usually the negative examples).

The steps of INDUCE are given below. These should be repeated in turn for multi-concept learning, with each concept becoming the target concept for each iteration of the algorithm.

1. For the target concept split the training examples into 2 subsets, a positive subset containing all examples belonging to the target concept and a negative subset containing all examples belonging to concepts other than the target concept.
2. Randomly select a single example from the positive subset to be used as a *seed* example
3. Take each knowledge item of this seed example in turn and find its coverage score
4. Order the knowledge items by 1) ascending negative coverage score and 2) descending positive coverage score . A knowledge item found in no negative examples and all positive examples is the most favourable. Include only those knowledge items within a pre-defined range (starting

from the best) in further processing. This ordered list is termed the partial star.

5. REPEAT
   (a) Remove solutions i.e knowledge items found in no negative and all positive examples. These solutions will be included in the concept ruleset.
   (b) Remove consistent knowledge items i.e knowledge items which are found in no negative examples, to a separate list of consistent knowledge items.
   (c) Each remaining knowledge item is specialised by combining it with knowledge items which occupy inferior positions in the partial star.

   UNTIL the number of desired consistent knowledge items is reached.
6. Each knowledge item on the Consistent list produced by step 5b is generalised by applying the following generalisation rules
   (a) The extension against rule
   (b) The closing interval rule
   (c) The climbing generalisation tree rule
7. Solutions are removed and added to the concept rule set.

4.3.3 RIGEL: Seed Selection and Look-Ahead Heuristic Search

The RIGEL Inductive Learning system (Gemello et al 1991) incorporates various heuristic strategies to optimise the hill-climbing search strategy of the STAR algorithm used by INDUCE. These strategies are 1) a priori evaluation of the positive example set to identify the best example to use as the seed and 2) a look ahead strategy to determine the best knowledge transformation strategy to use, according to the state of the current search space. These features distinguish RIGEL from learning systems that conform to a traditional algorithmic form (AQ, INDUCE, ID3 etc) in which a number of procedures

are executed in sequence and repeated until a goal has been reached. The heuristics that RIGEL uses to guide its search strategies are described using a *heuristic description language*. Each heuristic is represented as an expression linking members from a set of fuzzy linguistic variables that describe statistics relating to the state of the search space. Heuristics are applied to each formula of the partial star before any knowledge transformation is applied (RIGEL uses the VL_2 representation language: a *formula* in RIGEL is analogous to a *complex* in AQ). The statistics applicable to each formula include coverage, quantification, and the length of the formula. Examples of the linguistic variables which RIGEL uses are given in table 4.6.

variable	description
highpos	covers many positive examples
lowpos	covers few positive examples
lowneg	covers few negative examples
lowlength	contains few selectors
quant	is numerically or universally quantified
toogen	contains some selectors with many disjunctive values
noquant	is not quantified

Table 4.6: Linguistic variables used in RIGEL

Each linguistic variable is related to a real number in the interval {0,1}

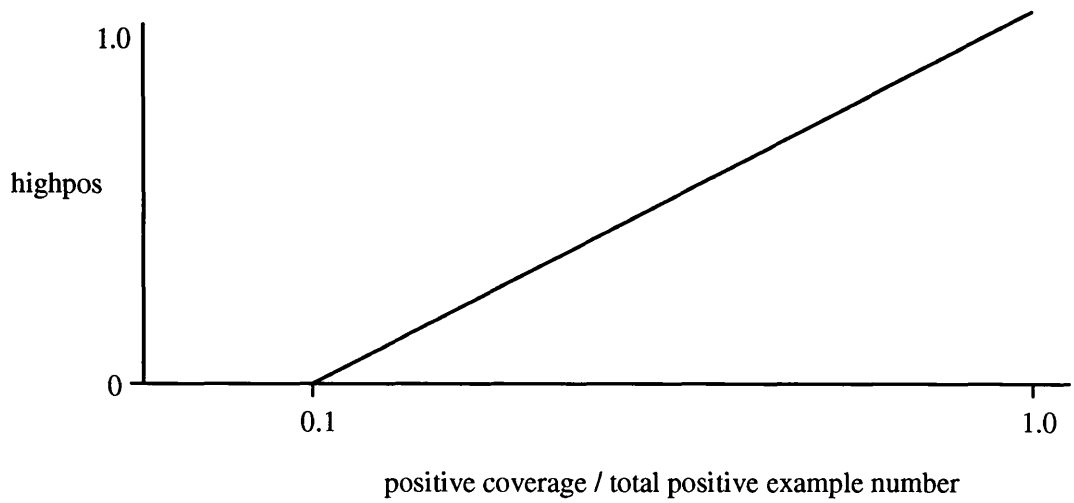

Figure 4.1: The relationship between highpos and positive coverage

that represents the "truth" value of the search statistic measured. For example, *highpos* describes how a ratio which measures positive coverage is related to the interval {0,1}. This ratio is the number of positive examples covered by a formula divided by the total number of positive examples. Figure 4.1 shows a more precise description of the truth value of highpos.

Heuristics are represented by expressions which are constructed by connecting these linguistic variables using (AND, OR and NOT)

Choosing The Best Example To Use As the Seed

Induction algorithms that use predicate logic as a knowledge representation language can be more prone to producing small disjuncts than those which use attribute-value pairs. This is because the application of specialisation and generalisation is limited to selectors found in only one example. It is impossible to widen the scope of the algorithm to apply these transformations

to all selectors in the positive examples (as in CN2 - see chapter 5) because of the problem of identifying whether a variable from a different example is the same as or different from a variable of the seed. Thus choosing a good seed from which to start the induction process can improve the overall classification accuracy of the induced ruleset. Choosing a poor (unrepresentative) example early in the inductive process can lead to a rule which covers not only that unrepresentative example, but a small number of more representative examples. The rule induced from the next iteration of the examples will also only cover a small number of examples. In this way a disjunctive cover can contain many small disjuncts. However, if each iteration of the algorithm uses the best seed present in the remaining uncovered examples then a more concise rule set will be obtained. The first rule induced will cover the most examples and rules induces subsequently will cover progressively fewer examples. The resulting ruleset will consist of a few large disjuncts describing the typical aspects of a concept and as few as possible small disjuncts describing how unrepresentative examples are exceptions to the typical rules.

The seed selection process used by RIGEL involves applying the following heuristic to each single selector in the positive examples:

(highpos AND (lowneg OR 0.9))

This heuristic favours selectors which cover many positive examples and few negative examples. The 0.9 in the right hand side lowers the score of selectors which cover many positive examples (highpos) and many negative examples (for which lowneg is 0). The selectors are sorted in descending order

from the selector that scores best according to the seed selection heuristic. Poor selectors are eliminated from consideration by a user defined threshold. (For example if the threshold was set at 0.3, any selectors scoring less than this would be removed.) The positive examples are evaluated by counting how many remaining (good) selectors are found in each example. The example containing the most is used as the seed. If there is more than one "best" example then the set of good selectors is split into subsets ordered according to the seed selection heuristic. Each subset contains all selectors with the same seed selection score. The "competing" examples are evaluated by comparison with the best subset and each succeeding subset in turn until one example scores more than all the others. If this process does not yield a single best example then the seed example is chosen at random from the remaining competitors.

Look-Ahead Search and Knowledge Transformation

When the seed example has been identified by the above procedure, RIGEL's look-ahead search strategy is applied initially to a partial star containing formulae which consist of a single selector from the seed. This step is also performed by INDUCE. However, whereas INDUCE specialises each formula until a set of formulae which cover no negative examples is found, and then generalises these formulae to extend the coverage of positive examples, RIGEL uses a number of heuristics to determine which knowledge transformation will produce the most promising results for each formula in the partial star. RIGEL also uses a different set of generalisation and specialisation operations

from INDUCE. There are 3 specialisation operations:

1. A selector is added to a formula if it contains a variable already present in the formula
2. A selector is added to a formula if no variables are shared
3. Quantification is applied to an unquantified formula by analysing how many times the individual selectors of the formula appear in the positive examples. This gives an indication of how many times a formula should appear in an unseen example. Formulae of this type conform to the following English statement: "The classification is positive if there exist between 3 and 5 black squares in the unseen example". This transformation is an example of constructive induction because the resulting formula is not present in the original representation.

There is one main generalisation operation which is described below. (The isolation of single selectors from the seed example is also a generalisation but it is not relevant to the look-ahead strategy)

1. The reference set of a selector is extented to incorporate those values of the domain background knowledge which do not appear in negative examples. Including only these values in the reference set allows this generalisation to avoid over-generalising the selector and thus covering negative examples. This operation can also be altered to include values which cover few negative examples, thus allowing the generation of probablistic rules which can deal with noisy data.

Each of the above operations has a condition of applicability associated with it. These conditions are represented using the heuristic description language described above. Each condition has a threshold, below which the operation will not be performed. Each formula in the partial star is evaluated to ascertain which operations should be performed. In this way new formulae are only constructed if they are more successful or simpler than the original formula.

The specialisation of formulae with no common variables is performed only if the selector to be added scores above the user defined threshold for the following heuristic:

(highpos AND lowlength)

Thus this specialisation is directed towards short formulae which cover a large number of positive examples. The quantification specialisation is controlled by a heuristic which prefers short formulae with no quantification which cover many positive examples:

(noquant AND lowlength AND highpos)

The generalisation strategy employed by RIGEL is directed to formulae which cover few positive examples and few negative examples. The values of the selectors in the formulae should also not be too general:

(lowpos AND lowneg AND (NOT toogen))

As in AQ and INDUCE, RIGEL has a beam search strategy which focuses the look-ahead heuristics on the best formulae in the partial star. Once all the new formulae generated by applying the appropriate operations have been added to the partial star, its contents are ordered according to the following heuristic:

(highpos AND lowlength AND (lowneg OR (lowlength AND (NOT quant))))

This heuristic favours short formulae which cover many positive examples and few negative examples or short unquantified formulae which cover many examples. A user-given threshold is used to remove the poorest formulae from the partial star.

The strategies outlined above allow the search to be focused on fewer, more suitable formulae than the less flexible search strategy used by INDUCE. As mentioned above the seed selection measure ensures that the individual selectors which represent the initial search state are the best possible. Thus not only does the seed selection measure ensure that the resulting ruleset contains as few small disjuncts a possible, but starting from the best initial state also optimises the search process.

4.4 Inductive Logic Programming and Theory Revision

Inductive Logic Programming (ILP) is the collective name given to a group of programs that use a predicate version of horn clause logic to learn rules from examples and background knowledge (Mooney and Califf 1995). The programs differ from the INDUCE family of induction algorithms by presenting the induced rules as logic programs (usually PROLOG), where variables present in the head of a clause (or rule) are linked to variables in the body of the clause, and the body of the clause is used to 'prove' the head of the clause. The head of the clause, as well as the components of the body are also termed *relations*. The relational nature of ILP is the reason why such programs have been used in the field of Knowledge Discovery in Databases (KDD). Indeed De Raedt and Bruynooghe (1992) argue that ILP and KDD are instances of a the more general problem of *belief updating*. This section will review FOIL, an ILP system which uses an information theory approach to relational learning and CLINT, a Multi-strategy theory revision system which combines ILP with abduction in an interactive learning environment.

4.4.1 FOIL

FOIL (Quinlan 1990) is an ILP system which integrates strategies used by both AQ and ID3 to solve the problem of learning horn clause rules from examples and previous definitions expressed as relational statements. The representation language is a subset of the PROLOG language and as such is also less expressive than VL_{21}. However the limitations imposed on this

representation allow the induction of rules with a recursive component, which is not possible with VL_{21}. Horn clause rules have only one clause on the LHS (consequent) and only allow the conjunction of clauses on the the RHS (antecedent). However clauses in the right hand side can be negated, for example:

$C \leftarrow D \wedge E \wedge \neg F \wedge \ldots$

(C is true if D and E are true and F is not true ...)

The above is an idealised propositional version of a horn clause, in the restricted form of PROLOG used by FOIL, clauses take the form of predicates (or relations) with variables. This gives the following definition of a FOIL rule:

$$P(X_1, X_2, \ldots, X_k) \leftarrow L_1, L_2, \ldots, L_n$$

where P is the target predicate with variables $(X_1, X_2, \ldots, X_k)$ and $L_1, L_2, \ldots, L_n$ are clauses which have been induced by FOIL to define the target predicate. (these clauses are termed *literals*). This rule would represent an *intensional* definition of predicate $P(X_1, X_2, \ldots, X_k)$: listing all the possible bindings for the variables would constitute an *extensional* definition of predicate $P(X_1, X_2, \ldots, X_k)$ (please see the illustrated learning situation below). Each literal of the RHS can be one of the four following forms:

$X_j = X_k$

$X_j \neq X_k$

$Q(V_1, V_2, \ldots, V_r)$

$\neg Q(V_1, V_2, \ldots, V_r)$

where the X_is are existing variables, the V_is are existing or new variables and Q is a relation. Variables in the predicates must contain at least one

existing variable. This restriction specifies that a newly induced literal should have a link to previous literals. This link will then ultimately connect with the variables in the target predicate.

The rule growing strategy of FOIL is similar to that used by the AQ family: literals are added to the RHS until the rule covers no negative examples of the target predicate. Remaining positive examples are covered by new rules with the same target predicate: thus different sections of the training set are covered by different rules. The training set for FOIL takes the form of extensional definitions of the target predicate split into two subsets: a positive subset for which instances of the predicate are true, and a negative subset containing false instances. Literals take the form of previous definitions for which there is a complete extensional definition given as background knowledge. The evaluation of candidate literals for adding to an incomplete rule is undertaken by an information heuristic which is similar to that used by ID3 to add new attributes to a decision tree. The target predicates of FOIL can only be defined in terms of true and false, thus like AQ, FOIL is a single concept learner.

An Illustration: Learning the Ancestor Relationship

Figure 4.2 displays a small family tree spanning four generations. This will be used to illustrate how FOIL builds a set of rules for the relationship:

ancestor(X,Y)

This is the target relation, that will appear as the left hand side of any induced rules. These induced rules should be capable of determining that

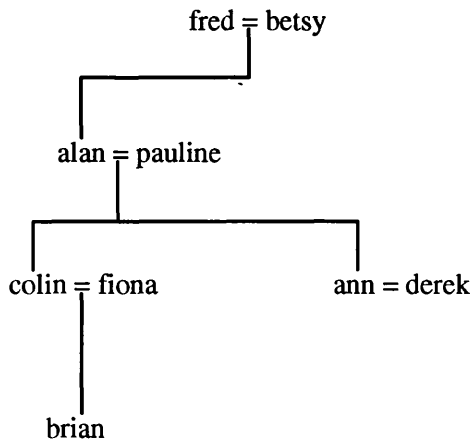

Figure 4.2: A simple family tree

both colin (father) and fred (great grandfather) are ancestors of brian. The starting position for FOIL is a training set containing negative and positive examples of the ancestor relationship (this is an extensional definition of ancestor(X,Y), the rules induced by FOIL would be an intensional definition) and a set of literals which are the previously defined clauses which will comprise the RHS of the ancestor rules. The extensional definition of ancestor(X,Y) is given below:

POS =
<fred,alan> <fred,colin> <fred,ann> <fred,brian>
<betsy,alan> <betsy,colin> <betsy,ann> <betsy,brian>
<alan,colin> <alan,ann> <alan,brian>
<pauline,colin> <pauline,ann> <pauline,brian>
<colin,brian>
<fiona,brian>

NEG =
<fred,fred> <fred,betsy> <fred,pauline> <fred,fiona> <fred,derek>
<betsy,betsy> <betsy,fred> <betsy,pauline> <betsy, fiona>
<betsy,derek>
<alan,alan> <alan,fred> <alan,betsy> <alan,pauline> <alan,fiona>
<alan,derek>
<pauline,pauline> <pauline,alan> <pauline,fred> <pauline,betsy>
<pauline,fiona> <pauline,derek>
<colin,colin> <colin,fred> <colin,betsy> <colin,alan>
<colin,pauline> <colin,fiona> <colin,ann> <colin,derek>
<fiona,fiona> <fiona,fred> <fiona,betsy> <fiona,alan>
<fiona,pauline> <fiona,colin> <fiona,ann> <fiona,derek>
<brain,brian> <brian,fred> <brian,betsy> <brian,alan>
<brian,pauline> <brian,colin> <brian,fiona> <brian,ann>
<brian,derek>

The literal 'parent(X,Y)' is also defined extensionally. It consists of the same pairs as ancestor(X,Y) but the distribution of positive and negative examples is different:

POS =
<fred,alan>
<betsy,alan>
<alan,colin> <alan,ann>
<pauline,colin> <pauline,ann>
<colin,brian>
<fiona,brian>

NEG =
<fred,fred> <fred,betsy> <fred,pauline> <fred,fiona> <fred,derek>
<fred,colin> <fred,ann> <fred,brian>
<betsy,betsy> <betsy,fred> <betsy,pauline> <betsy, fiona>
<betsy,derek> <betsy,colin> <betsy,ann> <betsy,brian>
<alan,alan> <alan,fred> <alan,betsy> <alan,pauline> <alan,fiona>
<alan,derek> <alan,brian>
<pauline,pauline> <pauline,alan> <pauline,fred> <pauline,betsy>
<pauline,fiona> <pauline,derek> <pauline,brian>
<colin,colin> <colin,fred> <colin,betsy> <colin,alan>
<colin,pauline> <colin,fiona> <colin,ann> <colin,derek>
<fiona,fiona> <fiona,fred> <fiona,betsy> <fiona,alan>
<fiona,pauline> <fiona,colin> <fiona,ann> <fiona,derek>
<brian,brian> <brian,fred> <brian,betsy> <brian,alan>
<brian,pauline> <brian,colin> <brian,fiona> <brian,ann>
<brian,derek>

The first step of the algorithm is to choose the best literal to add to the right hand side of the rule being induced. The best literal is that which has the highest information gain, the measurement of which is discussed in section 3.2. In this illustration there is only one literal (parent(X,Y) which can be chosen. However in a more thorough description of a family many other relationships may also be defined by literals, such as father, daughter, sister etc. If parent(X,Y) is added to the RHS of the rule, this first rule is complete, since there are no negative examples covered, However more literals are needed if the rule still covers negative examples. (This is analogous the specialisation step and rule stopping heuristic of AQ and INDUCE). Thus the first rule induced by FOIL for ancestor(X,Y) is:

ancestor(X,Y) :- parent(X,Y).

However this still leaves 8 positive examples uncovered:

<fred,colin> <fred,ann> <fred,brian>
<betsy,colin> <betsy,ann> <betsy,brian>
<alan,brian>
<pauline,brian>

To cover these, FOIL induces further rules, with the training set now consisting of the same negative examples and those positive examples not covered by the above rule. If the literal selection heuristic chose parent(X,Z), then the algorithm would expand the ancestor(X,Y) literal to give a set of triplets adding the third variable to uncovered pairs, giving a new training set, of which parent(Z,Y) covers:

POS =
<fred,colin,alan> <fred,ann,alan> <fred,brian,colin>
<betsy,colin,alan> <betsy,ann,alan> <betsy,brian,colin> <alan,brian,colin>
<pauline,brian,colin>
<fred,colin,pauline> <betsy,colin,pauline>
<alan,brian,fiona> <fred,ann,pauline>
<betsy,ann,pauline> <fred,brian,fiona>
<betsy,brian,fiona>
NEG =
<alan,alan,fred> <alan,alan,betsy> <colin,colin,alan>
<colin,colin,pauline> <ann,ann,alan> <ann,ann,pauline>
<brian,brian,colin> <brian,brian,fiona>

Each negative example is defined as such because the ancestor(X,Y) relationship it covers is in the negative subset of the extensional definition of ancestor(X,Y) (for example ancestor(alan,alan)), thus the rule:

ancestor(X,Y) :- parent(Z,Y).

needs further literals added. In this case the literal ancestor(X,Z) is added to produce the recursive rule:

ancestor(X,Y) :- parent(Z,Y),ancestor(X,Z).

This produces a further new training set:

POS =
<fred,colin,alan> <fred,ann,alan> <fred,brian,colin>
<betsy,colin,alan> <betsy,ann,alan> <betsy,brian,colin>
<alan,brian,colin>
<pauline,brian,colin>

which contains only positive triplets. These two rules combined cover all of the positive extensional definitions of ancestor(X,Y) and none of the negative ones. So the final intensional definition of ancestor(X,Y) is:

ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(Z,Y),ancestor(X,Z).

FOIDL - Inducing Decision Lists

FOIDL (First Order Induction of Decision Lists) (Mooney and Califf 1995) is a descendant of FOIL which addresses three shortcomings observed when FOIL was applied to the problem of learning the past tense of English verbs. These shortcomings are:

1. Background knowledge must be provided as a set of extensional definitions.

2. Explicit Negative examples of the target predicate must be included.
3. The clauses induced to describe the target predicate must be unordered and must not contain the cut (The cut is a PROLOG control operator which prevents the interpreter re-evaluating previous clauses).

The first shortcoming restricts FOIL's application to learning problems because the extensional definition of a background predicate can be infinite or large enough to be intractable. FOIDL uses background knowledge stored as intensional clauses, invoking the PROLOG interpreter when such a clause is being evaluated by the information heuristic. The variables of the example are used in this procedure to find out if the intensional clause satisfies the example. The second shortcoming is also one of intractability; the task of defining all of the possible examples of a target predicate may be infinite. This is overcome by a heuristic that estimates the negative coverage of a literal as a function of the unlinked variables giving the following equation:

$$\text{NEGCOV} = u^v - p$$

where u is an estimate of the 'realistic' number of extensional examples that may be encountered; v is the number of uninstantiated variables and p is the number of positive examples stored. FOIDL also stores a limited number of 'near miss' negative examples to further guide the induction process. Thus the search for good literals is guided towards literals with previously linked variables.

The third shortcoming is addressed by a slight change in the knowledge representation used to define the induced rule. First order decision lists are essentially PROLOG clauses in which the exceptions to the main rule

defining the target predicate are induced before the main rule itself: thus the induction process is changed to search for and cover exceptions first. The body of each induced clause is terminated by a cut, so that new cases representing exceptions would be classified first, leaving the more ordinary cases to be classified by a main rule which acts as a 'default' rule. This is a more direct way of describing the target predicate and can result in a less complex clause structure. Experimental results using the verbs problem described by mooney and Califf (1995) showed that the clauses induced by FOIDL were more accurate than those induced by FOIL.

4.4.2 CLINT - Interactive Theory Revision

CLINT (De Readt and Bruynooghe 1994) is an example of an Inductive Logic Program which uses induction to learn intensional predicates and abduction to learn extensional predicates. It also employs user interaction to verify its results at varying stages of the learning process. CLINT can also be regarded as an example of THEORY REVISION where an incomplete or incorrect theory is modified by varying inference strategies until it conforms to the target theory. The INTENDED INTERPRETATION contains the truth value for the CORRECT definitions of all predicates. Theory revision starts from the consideration of positive and negative examples of the target predicate. Classification of the examples into positive and negative examples is performed by outlining the truth values in the intended interpretation: positive examples are true; negative examples are false. The coverage concept is also used by CLINT, defined in the following way: 'an example e is covered by theory T if and only if e is logically implied by T' (De Readt and

Bruynooghe 1994). The interactive portion of CLINT consists of questions by which the user assists CLINT to revise a theory. Questions are of two types : 1) membership questions ask the user for the truth value of an example in the intended interpretation (the classification of an example); and 2) existential questions ask the user to provide the set of all variable instantiations of terms t_i in $p(t_1, \ldots, t_n)$ such that the resulting set of ground facts is true in the intended interpretation. Existential questions are generated by the abductive portion of CLINT.

The general problem solving strategy of CLINT can be defined as (De Raedt and Bruynooghe 1994):

Given:

A theory T containing definitions of intensional and/or extensional predicates

A set of positive and negative examples

A language definition L

A user willing to answer membership and existential questions about the intended interpretation

Find:

a revised theory T' obtained by adding/retracting facts for extensional predicates and adding/retracting clauses expressible in language L for intensional predicates such that T' covers all positive and no negative examples.

Two situations provide learning opportunities: a negative example covered by the current theory and a positive example not covered by the theory. In the former case, CLINT finds an incorrect clause and removes it; in the latter, CLINT uses induction and abduction to derive a number of clauses and/or facts which are added to the theory. In the first case, CLINT computes a proof tree and asks membership questions about the body of the clause that covers the negative example (with variables instantiated to the constants of the example). CLINT proves a clause to be incorrect if the head of the clause is false in the intended interpretation and the body is true.

A full description of the example domain used by De Raedt and Bruynooghe to explain the learning processes of CLINT is given below. The problem addressed is how to revise a theory concerning legal pairs of cards. The correct definitions are: a pair of cards is legal if one is a red number card and the other is a black face card; OR one of the cards is a joker (the other card can be anything). Three clauses are given (one of which is wrong !!)

1. legal(X,Y) → card(X) & card(Y) & red(X) & black(Y)

   (this clause is wrong, it is used to explain how CLINT deals with a covered negative example)
2. legal(X,joker) → card(X)
3. legal(X,Y) → black(X) & face(X) & red(Y) & number(Y)

This theory has correct definitions for the following predicates ...

card(X), red(X), black(X), diamonds(X), hearts(X), spades(X), clubs(X), face(X), number(X),
same_colour(X,Y), same_suit(X,Y), same_rank(X,Y), successor_rank(X,Y), lower_rank(X,Y).

The covered negative example situation is explained by assuming the system has encountered the example legal(diamond-7,spades-8) which is negative. Membership questions pertaining to the body of clause 1 would yield a value of true for both red(diamonds-7) and black(spades-8) making the body of the clause true and the head false; thus clause 1 would be removed. Positive examples which were covered by this clause are now reconsidered and those remaining uncovered are used to expand the remaining theory (abduction would add further ground facts, induction would add new clauses). CLINT attempts to find a correct theory which is closest to the original theory. To achieve this abduction is attempted before induction (new intensional clauses represent a larger deviation than new ground facts). The induction component of CLINT is described in the following section.

Using Induction to Incorporate An Uncovered Positive Example

The inductive process consists of two steps: 1) computing a most specific clause which covers the positive example and covers no negative examples and 2) generalising this clause by generating examples and asking the user membership questions. In this way CLINT can identify over-specific predicates and 'prune' them from the clause. These two learning situations demonstrate important characteristics of generalisation and specialisation

1. if a positive example is not covered by a clause no specialisation of that clause will cover that example.
2. If a negative example is covered by a clause all generalisations will cover the example.

The generalisation procedure used by CLINT is known as 'least general generalisation' and has an incremental nature. The process begins by considering the clause produced by step 1 above. The predicates belonging to this clause are analysed with respect to a new positive example. A new clause is formed with one predicate removed. The predicate removed is carefully chosen to cover the new positive example without covering any known negative examples. This process continues until no further generalisations can be found.

The resulting (maximally general) clause is then added to the theory.

Using the above example theory (legal card pairs) and assuming that the positive example legal(hearts-8,spades-queen) is uncovered, step 1 outlined above generates the following clause:

legal(X,Y) → hearts(X) & red(X) & card(X) & number(X) & spades(Y) & black(Y) & card(Y) & face(Y) & lower_rank(Y)

The next example to be considered might be legal(hearts-7,clubs-jack). In this case spades(Y) can be deleted. The generalised clause is:

legal(X,Y) → hearts(X) & red(X) & card(X) & number(X) & black(Y) & card(Y) & face(Y) & lower_rank(Y).

The next example might be legal(diamonds-7,clubs-jack). allowing hearts(X) to be deleted giving:

legal(X,Y) → red(X) & card(X) & number(X) & black(Y) & card(Y) & face(Y) & lower_rank(Y).

CLINT asks for truth verification for each example it uses. When it uses negative examples the predicate chosen for removal will result in a clause which covers that example, and therefore this predicate cannot be removed.

For example, legal(diamonds-7,hearts-jack) would allow black(Y) to be removed. This example is false so the intended generalisation would allow the new clause to cover false situations. Redundant predicates are also removed. For example, card(X) and card(Y) and lower-rank(X,Y), which are implied by face(Y) and number(X) are removed, giving the final maximally general clause as

legal(X,Y) $\rightarrow$ red(X) & number(X) & black(Y) & face(Y).

This is clause 3 of the example theory.

4.5 Version Spaces

Although Version Spaces may be considered to be independent of a representation language, they are included in this chapter because the concepts of coverage and generalisation/specialisation are relevant. They are also examples of single concept learners, in common with most algorithms described in this chapter. Version Spaces (Mitchell 1982, Hirsh 1992) treat the problem of acquiring a classifier as a search for the best classifier among the many possible which describe the training set of the problem. (In the case of version spaces the term 'classifier' is used to denote all of the descriptions used to define a single class) Indeed a version space is the set of all classifiers expressible in the description language that correctly classify the training set (Hirsh 1992). Incremental learning by the *candidate elimination algorithm* changes the version space by eliminating classifiers that do not correctly classify the new example. The goal of candidate elimination is the creation of a version

space containing only one classifier. This classifier is then considered the best one for classifying that particular problem.

Mitchell proves that the entire version space for a problem can be represented by 2 subsets: the S subset which contains the most specialised classifiers and the G subset which contains the most general classifiers. These subsets are the 'boundary sets' from which all other classifiers can be produced by various generalisation or specialisation steps. When a new example is introduced, the classifiers in both sets are tested on the new example. Those classifiers that give the wrong classification are removed and the two sets are updated. Classifiers from the S set can be minimally generalised to give a new set S$'$ which contains the next most specific set of classifiers. This procedure is carried out if removal of the classifiers would result in S being empty. A similar specialisation step can be applied to the most general set (G) when removal of unsuitable classifiers would also result in an empty set. In this way candidate elimination results in a convergence on a single best classifier. The [S,G] representation of a version space is used to overcome the NP-Complete nature of searching through all possible classifiers looking for the best one.

However Hirsh (1992) illustrates that the [S,G] representation of version spaces may itself contain exponential complexity. In particular, generation of G$'$ from G by specialisation can result in an exponential increase of classifiers in G$'$. He proposes a new representation [S,N] in which the G set of most general classifiers is replaced by a set N, which contains negative examples of the class. As before the set S contains the maximally specific classifiers which describe all positive examples and cover no negative examples (set

N). Candidate elimination proceeds by adding the new example to N if it is a negative example, and removing all members of S that do not classify it correctly. If the new example is positive S is generalised to give S', the set of minimal generalisations of S which cover no negative examples. The goal of this amended candidate elimination algorithm remains the same: create a version space containing only the best classifier.

4.6 Summary

From this chapter it can be seen that there are many different strategies and learning systems which induce rules from training examples. These systems can be categorised by the type of representation language used to describe both the training examples and the rules induced. Rule based induction algorithms have also been incorporated into multi-strategy systems in which a number of different learning strategies have been combined.

In common with decision tree induction, induction of rules from examples is a difficult process, often resulting in highly specific and inaccurate rulesets. The next chapter discusses the nature and cause of these problems, as well as outlining existing solutions in both the decision tree and rule based learning paradigms.

Chapter 5

Rule Complexity, Noisy Data and Pruning

5.1 Noisy Data and Complex Concept Representations

Training sets sampled from real world problems frequently contain data which complicate the machine learning task. In particular they contain 'noisy' data (Niblett and Bratko 1989) and/or data which conform to problems for which the knowledge representation of the data is inadequate (Callan and Utgoff 1991). Noise represents data that have been collected or recorded wrongly and in which the errors can take two forms: 1) random variations in the attribute value preconditions of the data and 2) misclassifications where the class membership has been altered. An additional complication presented by these errors is the presence of duplicate data where two training examples share the same pre-condition attribute values but differ in class membership. Inadequate knowledge representation occurs when the language used to encode the training examples cannot express the most important aspects of the

learning problem. The first sections of this chapter refer to induction of rules and trees from training sets represented using propositional logic in which noise has a detrimental effect on the learning process. Inadequate knowledge representation is described in a section 5.4 which also discusses constructive induction.

5.2 Noise, Simplicity and Classification Accuracy

Many real world problems (Holte 1993) have simple underlying structures which, in theory, should allow machine learning systems to describe their target concepts in a succinct form. In fact many concept subsets (examples belonging to a single classification) consist of a small number of groups containing similar examples, each of which should be correctly described by a single short rule in rule induction systems or by a single short branch in decision tree induction. However, the presence of misclassified training examples obscures this simple structure giving the impression that each single concept subset contains a large number of disimilar examples, many of which are more similar to examples belonging to one of the other concept subsets. This added complexity in the training set results in very complex representations of target concepts being produced by programs which belong to both the decision tree and decision rule learning paradigms. These complex representations induced from noisy training sets are often poorer classifiers when tested on the whole population. There has been much previous research devoted to developing noise tolerant learning systems and also in the related

topic of what measurements should be used to determine good classifiers.

Occam's Razor has been used for guiding the development of decision tree and rule based inductive learning systems for many years. It was originally formulated as a guide for general scientific discovery: 'the simplest theory devised to explain an observation is usually the best'. A modern version of Occam's Razor, from a machine learning perspective, is given by Webb (1996)

> "Given a choice between two plausible classifiers that perform identically on the training set, the simpler classifier is expected to classify correctly more objects outside the training set"

Research that has been devoted to testing the applicability of Occam's Razor in decision tree induction has usually concentrated on noise-free training data. Murphy and Pazzani (1994) induced a 'forest' of decision trees of various sizes, each of which correctly described the training data. They compared the average classification accuracies of groups of decision trees of increasing size and found that the best classifiers were those trees which were slightly larger than the smallest tree. Decision tree size was measured by counting the number of nodes in the decision tree. This result casts some doubt on the automatic acceptance of Occam's Razor as the most favourable guide for inductive learning.

However Fayyad and Irani (1990) suggest that classification accuracy and the overall performance of classifiers is related more to the number of branches (or rules) found in the classifier. More particularly, they found that decision trees in which each branch covered a large proportion of the

training set correctly would be more successful classifiers than decision trees that contained a larger number of branches each covering fewer members of the training set. They thus identify branch number as a good predictor of error rate (assuming that the decision trees induced covered the training set correctly). A large proportion of the branches or rules comprising a complex ruleset or decision tree will contain a large number of pre-conditions. This is because the specialisation operation which is necessary to achieve correct coverage adds further conditions and so restricts the coverage of the branch (rule) to fewer training examples, and thus to fewer test examples. More rules with more conditions are thus needed to correctly classify the entire training set. The induction of complex rulesets occurs when training datasets are noisy or represent learning problems with representational complexity as outlined in the previous section. The average number of conditions per rule is thus also lower for rulesets with fewer rules. The lower classification accuracy of more complex rulesets containing rules which cover fewer training examples is described as the 'small disjuncts' problem by Holte et al (1989).

In a further study of rule simplicity, Holte (1993) has investigated the classification accuracy of very simple rules designed to capture the simple underlying structure of real world data. He compares the performance of a new system 1R with C4.5 on a number of well known machine learning data sets. 1R is a learning system which uses error rate to establish which attribute would be the best to use as a one-level decision tree (1-rule). The resulting rulesets are thus very simple and in 12 out of the 14 data sets used for comparison, the resulting 1-rules had a classification accuracy of only 3.1 percent less than the more elaborate decision trees produced by C4.5. The

1-rules performed significantly worse than C4.5 on the other 2 datasets used in the comparison. Holte has used a number of measures to determine the complexity of the decision trees produced by C4.5. These are defined below.

1. Maximum depth — this is the maximum depth of the (pruned) decision tree. It may not be an accurate measure of the true complexity of the tree, because complex trees may be wide as well as deep.
2. Dynamic complexity — this the average number of attributes used for classifying an example.

Using both measures, decision trees produced by C4.5 from all datasets were significantly more complex than the 1-rules produced by 1R. Holte's report illustrates that very simple rules can achieve classification accuracies which are close to those achieved by more sophisticated induction systems. Holte uses this result to justify his claim that real world data is inherently simple, and that complexity in the form of noise can result in sophisticated induction algorithms obscuring this simple structure.

5.3 Methods for the Simplification of Complex Rulesets

Existing methods for producing simple rulesets from noisy data use the following strategies:

1. Pre-Pruning: Significance testing and error estimation are used to terminate rule growing before complex rules are produced.

2. Post-Pruning: "Undesirable" rules and conditions are removed from complex rulesets.

3. Altering Stopping Criteria: The preference for maximally general rules is relaxed so that the error rates of small disjuncts is decreased.

4. Induction of Rules with a component that includes probablistic estimate of classification accuracy.

5.3.1 Pre-Pruning of Rules: The CN2 algorithm

The first strategy is used by CN2 (Clark and Niblett 1988). The CN2 algorithm modifies the basic AQR algorithm to remove its dependence on specific training examples. (AQR uses a seed example and then specialises by comparing complexes with various positive and negative examples). Removing this dependence allows CN2 to start from the best complexes found in the entire training set and so induce the rule with highest coverage at each stage. Starting from a seed example restricts the choice of attribute value pairs and so could lead to the construction of rules that cover fewer examples. The removal of the seed example thus allows CN2 to construct less complex rules. CN2 can induce rules from noisy situations by considering rules which cover a small number of negative examples (rather than no negative examples which is the case with AQ). In this way CN2 handles noise by pre-pruning; the algorithm contains heuristics to terminate the rule growing process before rules containing many conditions are induced. The specialisation process of CN2 also appends selectors from the entire representation space rather than from a specific example (as for AQ). The bounds of the search are delim-

ited by only keeping the best complexes at each specialisation step. To find the best complex(es), the information gain heuristic used in ID3 is applied to the probability distributions of the classifications of examples covered by each complex. This is in contrast to AQ, in which a simple coverage score is used as a measure of the suitability of complexes. This heuristic favours complexes which cover a large number of examples from a single class and few examples of other classes.

CN2 also tests complexes for significance by using the likelihood ratio statistic to test if the complex represents a genuine correlation between the selectors and the classification. The set of best complexes are those which are both significant and score best according to the information heuristic. The steps of CN2 are given in table 5.1.

Let E be a set of classified examples
Let SELECTORS be the set of all possible selectors

Procedure	CN2(E)	
Let	RULELIST	be the empty list
Repeat	until	BEST_CPX is nil or E is empty
	Let BEST_CPX	be Find_Best_Complex(E).
	If BEST_CPX	is not nil.
	then	let E′ be the examples covered by BEST_CPX.
		Remove from E the examples E′ covered by BEST_CPX.
		Let C be the most common class of examples in E′
		Add the rule 'if BEST_CPX the class is C' to the end of RULELIST
Return	RULELIST	

Procedure	Find_Best_Complex(E)
Let	STAR be the set containing the empty complex
Let	BEST_CPX be nil
While	STAR is not empty,
	Specialize all complexes in STAR as follows:
	Let NEWSTAR be the set $\{x \wedge y \mid x \in STAR, y \in SELECTORS\}$
	Remove all complexes in NEWSTAR that are either in NEWSTAR (i.e the unspecialized ones) or null (eg big = yes & big = no)
	For every complex C_i in NEWSTAR:
	if C_i is statistically significant and better than BEST_CPX by user defined criteria when tested on E
	Then replace the current value of BEST_CPX by C_i
	Repeat Until the size of NEWSTAR $=<$ *maxstar* (A user defined maximum)
Remove	the worst complex from NEWSTAR
Let	STAR be NEWSTAR.

Table 5.1: Steps of the CN2 algorithm

However, Holte et al (1989) have shown that the statistical techniques used by CN2 also eliminate all small disjuncts, thus the resulting rulesets will be poor if the training data contains a significant number of atypical examples. They further report that a system which uses both significance testing and an error estimation technique such as the Laplacian error estimate would be more successful at removing error-prone small disjuncts.

5.3.2 Post-Pruning: Truncation of Covers and Reduced Error Pruning

The truncation procedure (TRUNC) of AQ15 (Michalski et al 1986) is an example of the second strategy. It removes very specific rules by identifying and removing rules which cover few examples. This is analogous to the tree-pruning strategies used by C4.5 (Quinlan 1993), CART (Brieman et al 1984), and ASSISTANT (Cestnik et al 1987). The procedure TRUNC uses two measures t and u. The t measure represents the total number of examples explained by a complex and the u measure represents the total number of examples *uniquely* explained by a complex. These two measures quantitatively describe the effects of removing complexes from a cover. The u measure of a complex that is removed from a cover represents the number of examples that the cover no longer explains. TRUNC sorts the complexes of a cover according to the t measure and removes the complex with the lowest t and u score. The authors (Michalski et al 1986) claim that noisy examples will produce complexes with low u scores and that removing these reduces the complexity of the cover whilst retaining as much accuracy as possible. Accuracy is further improved by applying a flexible evaluation

scheme to the recognition rules produced from the truncated covers. This scheme chooses the most probable match when a new example either 1) matches no recognition rule or 2) matches two or more recognition rules.

Another similar approach, Reduced Error Pruning (Bunk and Pazzani 1991) partitions the training data into two subsets, in which the primary training set is used to induce a complex ruleset which is correct and the retraining set is used to test subsets of the initial ruleset. These subsets are obtained by applying simplification operations such as rule removal and rule condition removal systematically. The subset which has the lowest error on the retraining set is chosen as the best ruleset and replaces the initial complex ruleset. However, Cohen (1992) has found that the computation time required to conduct reduced error pruning has an exponential relationship with problem size for training sets containing 10% noise. He suggests 'reduced error rule set regrowth' as an alternative. This strategy also begins by inducing a complex ruleset from the primary training data. This ruleset is simplified by removing conditions and the resulting set of generalised rules which has the least additional error on the primary training set is added to the initial complex ruleset. The pruned ruleset is obtained by adding rules from this larger ruleset to an initially empty ruleset if they reduce the error on the retraining set. This process is computationally much less costly than reduced error pruning. Such an exploration of the possible generalisations of a ruleset is similar to the concept of a *Set Enumeration* (SE) Tree (Rymon 1993). An SE tree represents a generalisation of a decision tree by containing more than one attribute at each node. Induction of SE trees is achieved by using the splitting criterion to rank the attributes at each node, thus the

root node has expansions representing all attribute value pairs. This procedure also organises the SE tree so that the left side of the tree represents the normal decision tree which would be induced by applying the splitting criterion in the usual manner for decision tree induction. Hill climbing search procedures can be applied to SE trees to find that portion of the tree which has least error on noisy training data (Rymon 1996).

5.3.3 Altering the Stopping Criteria for Rule Growth

Holte et al (1989) prefer the third approach to simplifying complex rulesets. Their approach uses a strategy that examines the coverage of each rule induced by CN2 to identify small disjuncts, defined as those rules with a coverage below a pre-defined threshold. The stopping criteria for these rules are changed to allow small disjuncts to become maximally specific i.e. a small disjunct is specialised further by adding more conditions to reduce its coverage still further. Rules found to cover many examples remain unchanged. The resulting ruleset will contain a number of large disjuncts covering the majority of the training set and a large number of very specific rules which are each specific to a very small number of atypical examples. However, they found that applying the maximally specific rule selection criterion to small disjuncts had the effect of reducing the accuracy of larger disjuncts. To overcome this they propose a selective specialisation criterion which finds the optimum specialisation for small disjuncts, thereby minimizing the error rate of the ruleset as a whole. The selective specialisation criterion sets a coverage limit for each attribute value pair of the rule. A condition will only be added to a small disjunct if it covers less than 25% of examples which do

not belong to the target classification.

5.3.4 Inducing Probablistic Rules and Rules with Flexible Matching

HYDRA: Inducing Probablistic Rules

The fourth strategy is implemented in a relational rule learning system called HYDRA, developed by Ali and Pazzani (1993a). HYDRA is a descendant of the FOIL system developed by Quinlan (1990) discussed in chapter 4. Ali and Pazzani have found that FOIL suffers from the small disjuncts problem (Ali and Pazzani 1993b). This is because it has a similar separate and conquer strategy to CN2 where large rules are found first and the covered examples removed, leaving the 'awkward' examples to be covered by complex rules. HYDRA learns probablistic rules which incorporate a likelihood ratio as part of the concept description. This likelihood ratio provides a combined measure of the negative and positive coverage of the concept description. This ratio is used in place of the information heuristic used by FOIL so that rules can be learned which have a probability associated with how correctly they describe the training set. Probablistic rules require shorter rules because the strict coverage conditions have been relaxed. The resulting rulesets are more concise, containing fewer small disjuncts.

POSEIDON: Flexible Matching

This algorithm uses concepts acquired by the AQ15 algorithm as its starting point. Concept descriptions acquired by AQ15 are inflexible; a new example must totally match the recognition rule or it does not belong to the concept.

POSEIDON (Bergedano et al 1992) attempts to overcome this inflexibility by introducing *flexible concepts* which allow less precise matching of unseen examples. This is achieved by measuring the degree of match between the concept and the new example and constructing inference rules that can alter the concept description to take into account the context in which the unseen example is used, and the typicality of the unseen example.

Bergedano et al (1992) argue that these methods produce a concept description which is more akin to how humans perceive concepts by making the concept boundary imprecise and by taking context into account in any application of the concept. The flexible matching function (the degree of matching measure mentioned above) and the context dependent inference rules can allow concept recognition rules to recognise unseen examples with missing and incorrect attribute values.

Flexibility of concepts is realised using a "Two Tiered Representation". This representation uses the VL_1 language developed for the AQ series, but in POSEIDON, concepts have two components (or "tiers"), as opposed to the single component found in AQ. These two components are referred to as the Base Concept Representation (BCR) and the Inferential Concept Interpretation (ICI). The BCR consists of the basic features which are common to most examples of the concept, and the ICI contains the flexible matching function and the inference rules which can transform the BCR into a description which can deal with non-representative, or atypical examples.

The BCR describes the "central tendency, the most relevant properties and the intention" (Bergedano et al 1992) of a concept description. This information may be represented in many forms, including a complete and

consistent set of concept recognition rules acquired by AQ, a proof tree supplied by explanation based learning, and a set of cases acquired by Case Based Reasoning. This rigid description is altered by the ICI to give an extended description (concept extension), or a more specialised description (concept contraction).

Using the two tiered description involves matching an unseen example with the BCR and measuring the result. The inference rules contained in the ICI may be able to extend or contract the BCR to fit the example, if the example is atypical.

The Flexible Matching Function F, matches events (selectors from the new examples) and expressions from the concept descriptions to the interval [0..1]. This match is obtained by matching an event with each rule which makes up a concept description. This is expressed as the average of the degrees of fit for its constituent conditions, weighted by the proportion of positive examples to all examples covered by the rule. The equation is given below:

$$F(e,r) = (\sum F(e,c/n)) \times \#rpos/(\#rpos + \#rneg)$$

where F(e,c) is the degree of match between event e and condition c in rule r, n is the number of conditions in r, and #rpos and #rneg are the number of positive and negative examples covered by r respectively.

The degree of match for each condition F(e,c) is dependent on the type of expression making up the condition. The four possible types are described

Type	Description
Nominal	an unordered domain
Linear	a domain ordered by some successor relationship
Structured Nominal	an unordered generalisation tree
Structured Linear	a generalisation tree ordered by some successor relationship

Table 5.2: Description of the types of expressions used by POSEIDON

in table 5.2.

For nominal and structured nominal expressions; if the unseen example satisfies the expression then F(e,c) is 1, otherwise F(e,c) = 0. For the linear and structured linear expressions: these expressions can take a range of values. If the value of unseen example expression falls within this range, then F(e,c) is 1, if the value of the unseen example expression does not fall within this range then the distance between the end of the range and the example value is used to calculate F(e,c) . The Flexible Matching Function is set to zero or "no match" if the degree of match is lower than a predetermined threshold.

The Flexible Matching Function is useful for dealing with examples which are close to the typical case, whereas the inference rules, also contained in the ICI tier, contain the transformations necessary to deal with exceptions to the typical case, or the context in which the concept description is used.

An event can have three types of match, depending on which part of the two tiered representation is used to perform the match.

1. If an event matches the BCR then it is S covered (A strict match)

2. If the flexible matching function is used to match an example to the BCR, then the event is F-covered

3. If the deductive inference rules transform the example into a match, then the example is said to be D-covered.

These types of match can be said to be a measure of how representative an example is of a concept: S-covered examples are representative, F-covered examples are nearly representative, and D-covered examples are exceptions.

5.4 Representational Complexity and Constructive Induction

However, as mentioned above, misclassification is not the only cause of the complexity found in learning problems. This section deals with complexity which is found in training sets containing no errors and no noise. This complexity is representational in nature and relates to the character of the problems themselves. However it is also common for learning problems to exhibit both kinds of complexity.

The AQ17-HCI (Wnek and Michalski 1994a) inductive learning system alters the basic AQR algorithm to take account of Constructive Induction (HCI stands for *Hypothesis-driven Constructive Induction*). Constructive Induction is used to derive new attributes which may describe the decision making domain better than the attributes which have been used to describe the examples. This represents one solution to the problem of using a propositional logic language to describe domains which would be more effectively described using a predicate logic language capable of representing examples

as structured objects (see section 4.3 which describes predicate logic induction). In these situations propositional induction systems (both rule based and tree based) produce large rule sets which have low classification accuracy. Michie et al (1994) describe how constructive induction was applied to a problem in the chess domain. Initially, a decision tree learning system was used to construct a tree of 27 nodes which had a classification accuracy of 69 % (this is close to the accuracy achieved by "guessing" by choosing the most likely classification for each unseen example). Constructive induction was used by augmenting the original six attributes with 15 new attributes. The decision tree produced from these 21 attributes achieved 99% accuracy but it was very cumbersome with 49 nodes. The Inductive Logic Program GOLEM (Muggleton and Feng 1990) achieved 100% accuracy and produced only 4 rules.

The steps in the AQ17-HCI algorithm are given below (Wnek and Michalski 1994b). The positive subset is represented by the symbol E^+ and the negative set is represented by E^-:

1. Divide randomly each of the training sets E^+ and E^- into two subsets: the primary set E_p and the secondary set E_s. $E^+ = E_p^+ \cup E_s^+$. $E^- = E_p^- \cup E_s^-$. The primary training subset is to be used for rule generation, and the secondary subset is to be used for rule verification.
2. For each concept, induce the most specific cover of the set E_p^+ against the set E_p^-.
3. Evaluate the performance of the rules on the secondary training set, E_s. If the performance exceeds a pre-defined threshold then go to step 8.
4. Analyse the rules to identify possible changes in the representation space.
5. Change the representation space by removing irrelevant attribute values or attributes, or by adding new attribute values or attributes.

6. Modify the training set of examples, E, according to the changes in the representation space.
7. Go to Step two.
8. For each concept induce a set of the most specific rules from all positive examples against all negative examples, i.e. a cover COV_{ms} (E^+/E^-) and the most general cover of negative examples against positive examples COV_{mg} (E^-/E^+).
9. Build final concept descriptions by generalising the most specific positive rules against the most general negative rules, i.e.
   COV_{mg}(COV_{ms} (E^+/E^-) / COV_{mg} (E^-/E^+)).

The last step produces the most general positive rules by removing as many selectors from the positive rules as possible. This process continues until any further removal would result in coverage of a negative rule.

Chapters 6 and 7 outline an alternative method for inducing simple classifiers. Experimental data is also used to show that classifiers induced by this method can often obtain similar or higher classification accuracies than some of the methods outlined above. These classifiers do not correctly classify the all of the training set, but attempt to trade-off poorer training set accuracy for increased classification accuracy on the test set.

Chapter 6

Alternative Methods for Inducing Simple Rules

In this chapter a new approach to the problem of inducing simple classifiers is described. It differs from those described in the previous chapter by concentrating on identifying difficulties in the training set itself rather than concentrating on either the methods for discovering rules or on altering the rules induced from the training set. Identifying those training examples which present problems for machine learning systems opens up opportunities which can improve the quality of the resulting ruleset. If the problem examples represent misclassifications within the training set, identifying these examples and presenting them to the expert for correction will allow induction to proceed from a correct training set. Identifying these examples can also allow learning to focus on easy representative examples. The problem examples may then be handled by a different learning paradigm. This chapter will explore these issues in more detail.

The strategies central to this approach are related to the procedure used

by RIGEL (Gemello et al 1992) to choose the best training example to seed the induction process. As described in chapter 4, this procedure scores the selectors comprising an example by determining how common they are in the positive examples and how rare they are in the negative examples. The most favourable selector is that which is in the largest number of positive examples and also in the fewest number of negative examples. The best example to use as the seed contains the largest number of favourable selectors.

This procedure is successful in choosing a good seed example because it finds the example most representative of the positive class. The strategies presented in this thesis modify this approach so that the least representative examples of a class are found. These will be those examples which score poorly when scored by RIGEL's seed selection procedure. Although RIGEL is a learning system which employs predicate logic, the strategies presented here can only function with propositional learning systems such as CN2 and C4.5. Extending the strategies to work with predicate logic systems will be discussed in Chapter 8 (as future work).

This chapter presents a new strategy (known as *CN2-D*) which has the aim of determining the best way to identify unrepresentative or *difficult* examples. The further sections of this chapter describe this strategy, its development, the results obtained from comparing the new strategy to an existing strategy (CN2), and a modification of the strategy appropriate for decision tree learning. In particular, section 6.2 describes the evaluation methods used in the experiments described, section 6.3 describes experiments undertaken in the development stages of CN2-D, section 6.4 describes experiments which compare the performance of CN2-D to CN2, and section 6.5 describes

results obtained by applying a variant of the strategy to C4.5. The strategies and algorithms are compared by inducing rulesets from a number of problems which reveal differing characteristics. For all experiments, performance is evaluated by measuring both the ruleset complexity and the classification accuracy of the rulesets induced.

6.1 CN2-D: A Strategy for Identifying Difficult Training Examples

The first stage of CN2-D involves obtaining a score of how representative each training example is of its class. This is achieved by combining the relative frequency of the individual attribute values which describe the example. This score is termed the *d-score*.

The relative frequency for an attribute value for a given class can be calculated by following equation. Given n examples of a given class, c, and r occurrences of a particular attribute value, v, within these n examples, then the relative frequency of observing attribute value v, for a member of class c is:

$$F_{vc} = p(value = v|class = c) = \frac{r}{n}$$

With the relative frequency calculated, the d-score for an example, e, for a given class c is given by the following. For example e, attribute values are

$v_{ei}, \ldots, v_{em}$ where there are m attributes:

$$d - score = \frac{1}{m} \sum_{i=1}^{m} F_{v_{e_i},c}$$

The d-scores are divided by the number of attributes so that comparison between different training data populations is possible.

The d-score is repeated for each possible classification of the example to give a tuple which contains as many classification/d-score pairs as there are possible classifications in the problem. The classification/d-score pairs comprising the tuple for each example are used to determine which examples are unrepresentative. Unrepresentative examples identified by this strategy will be termed *difficult.* where difficult is defined as follows:

> *An example is described as difficult if the classification/d-score pair with the highest d-score has a classification other than the actual classification of the example.*

After CN2-D has identified the difficult examples, a list of these examples can then be checked by a human expert. If any examples are unrepresentative because they have been misclassified, the human expert can correct these and induction can proceed from a more reliable training set. (In the animals example described below the unrepresentative examples identified by the d-score are misclassifications which could easily be corrected). In this way CN2-D can be used interactively, allowing corrections to be made before induction proceeds.

example No	class	Attribute				d-score(CN2-D)	
		covering	legs	wings	colour	bunny	owl
1	bunny	fur	4	no	grey	0.3125	0.5830
2	bunny	feathers	2	yes	black	0.5000	0.3333
3	bunny	fur	4	no	brown	0.3125	0.5830
4	owl	feathers	2	yes	grey	0.5625	0.3333
5	owl	feathers	4	yes	brown	0.5625	0.3333
6	owl	feathers	2	yes	white	0.6250	0.2499
7	owl	fur	4	no	white	0.3750	0.5000

Table 6.1: A small training set describing owls and bunnies

More particularly, CN2-D has been used to pre-process training data sets which are then presented to the CN2 algorithm. Examples identified as unrepresentative or difficult are removed and CN2 only induces rules from the reduced 'easy' training sets. The training set for a small problem involving animals is given in table 6.1.

The d-scores obtained by CN2-D (together with those obtained by CN2-U, see section 6.2.2) are also given for each example. This problem will be used to illustrate how CN2-D identifies difficult examples and how removal of these examples affects the induction of rules. Table 6.2 contains the relative frequencies for the attribute-value/class combinations found in the training set illustrated in table 6.1.

Tables 6.3 and 6.4 display the d-scores obtained for examples 1 and 2

Attribute	Value	Class	
		Owl	Bunny
Covering	Fur	1/4	2/3
	Feathers	3/4	1/3
Legs	Two	2/4	1/3
	Four	2/4	2/3
Wings	Yes	3/4	1/3
	No	1/4	2/3
Colour	Grey	1/4	1/3
	Brown	1/4	1/3
	White	2/4	0/3
	Black	0/4	1/3

Table 6.2: Relative frequencies for attribute-value/class combinations

respectively.

The examples which belong to this problem are described by 4 attributes and can belong to one of two possible classes (owl or bunny). The tuple obtained by CN2-D for each example will thus have two d-scores, one for class *owl* and one for class *bunny*.

In this training set, CN2-D would identify examples 2 and 7 as difficult. All the rest would be considered 'easy'. Bunny number 2 and Owl number 7 are probably misclassifications; a human observer with prior experience of bunnies and owls would be surprised to find a bunny which had feathers and

class	attribute value				d-score
	fur	four	no	grey	
class if owl	(1/4 +	2/4 +	1/4 +	1/4)/4	0.3125
class if bunny	(2/3 +	2/3 +	2/3 +	1/3)/4	0.5830

Table 6.3: d- scores for example no 1

class	attribute value				d-score
	feathers	two	yes	brown	
class if owl	(3/4 +	2/4 +	3/4 +	0/4)/4	0.5000
class if bunny	(1/3 +	1/3 +	1/3 +	1/3)/4	0.3125

Table 6.4: d-scores for example no 2

flew.

6.2 Evaluation Methods

6.2.1 Learning Problems

The performance of each algorithm has been compared by applying each algorithm to a number of learning problems. All experiments are carried out on the same five learning problems. The data sets corresponding to each problem are all artificial, but each has been chosen to illustrate the various aspects of difficulty described above. The contact lens problem (Cendrowska 1987) is a complete set, easily described in the attribute value pair represen-

tation, with no misclassifications and a low proportion of difficult examples. The mushroom problem is an incomplete data set manufactured by the author which is not described well by the 7 attributes used. There are a number of identical mushrooms belonging to different classifications which would require additional attributes to distinguish them. The Monks problems were designed by Thrun et al (1991) as artificial data sets for an unbiased comparison of the performance of learning algorithms from a number of different paradigms. These included neural networks, induction algorithms and clustering algorithms such as COBWEB. The Monks problems are all concerned with an artificial robot domain, in which the robots are described by 6 discretely-valued attributes. The total population size was 432 examples for each problem and each problem has a classification attribute with two values. The different problems each have different characteristics. Monks 1 is easily described in the attribute-value pair representation and contains little difficulty. Monks 2 contains a significantly larger number of difficult examples because it was created to illustrate how the classification accuracy of induction algorithms which use only the attribute-value representation decreases when they are presented with problems which are difficult to represent in this language. Monks 3 has a small amount of difficulty which has been introduced to illustrate the performance of algorithms in noisy situations where training examples are misclassified.

6.2.2 Test Conditions

Two different sets of test conditions were applied to the problems in the following ways. The first set of conditions was applied to the mushrooms

and contact lens problems. The data set represented by each problem was randomly split into 50% for training and 50% for testing. A larger proportion of the populations representing the mushrooms and contact lenses problems was used for training because these problems contain many fewer examples than the Monks problems. The easier training conditions were offset by a tougher test regime where no test example had been part of the training set. The test conditions used for the Monks problems were the same as those used in the original comparison by Thrun et al (1991), namely the test set contains all possible examples. Monks 1 has a training set of 124 randomly selected examples, Monks 2 has a randomly selected training set of 169 examples. Monks 3 has 122 randomly selected training examples, of which 5% have had their classification altered to give a noisy training set.

6.2.3 Evaluation Measures

Ruleset complexity was evaluated by a measure which describes the average number of conditions per class for a given ruleset. It is obtained by dividing the total number of conditions (attribute value pairs found in the ruleset) by the number of classifications found in the problem.

$$ruleset\ \ complexity = \frac{no\ of\ attribute\ value\ pairs}{no\ of\ classes}$$

This measure gives a good indication of the number of specialisation steps needed for CN2 to classify one class in the training set. It is averaged over all classes so that rulesets from different learning problems can be compared. The following simple ruleset has a ruleset complexity of 5 conditions per class.

$$\text{class} = 1 \quad \rightarrow \text{a} = 1 \;\&\; \text{b} = 1$$

$$\text{class} = 1 \quad \rightarrow \text{a} = 1 \;\&\; \text{b} = 2$$

$$\text{class} = 2 \quad \rightarrow \text{a} = 2 \;\&\; \text{b} = 3 \;\&\; \text{c} = 2$$

$$\text{class} = 2 \quad \rightarrow \text{a} = 2 \;\&\; \text{b} = 3 \;\&\; \text{c} = 1$$

The classification accuracy of the rules produced by CN2 and CN2-D was evaluated by the following strategy. A 'correctness' score was produced for each test example. The correctness score is 1 if all the rules which cover the example are from the correct class. Otherwise correctness is measured by the following equation:

$$correctness = \frac{no\ of\ rules\ from\ the\ correct\ class}{total\ number\ of\ rules\ covering\ the\ example}$$

Classification Accuracy is then the sum of the correctness of each example divided by the total number of examples.

6.3 Experiments Concerning the Development of CN2-D

6.3.1 Experiment1: Comparing CN2-D to CN2-U: A Simpler Strategy For Evaluating Training Sets

In this section, the performance of CN2-D is compared with that of CN2-U, a simpler strategy based on the identification of training examples which are unrepresentative of their class. Examples identified as unrepresentative by CN2-U are simply less like other members of their class. This is in contrast to CN2-D where difficult examples are less like their own class *and* more like members of other classes. CN2-U was developed by the author and uses the

d-score in a simpler way which is closer to the original seed selection method used by RIGEL. In contrast to CN2-D, CN2-U only generates one d-score for each example, i.e CN2-U only evaluates examples by how they score for the class they belong to, rather than all classes in the learning problem. When every example been scored the scores for the examples belonging to a single class are *scaled* in the range {0:1}: i.e. the highest scoring example for that class is given the value 1 and all other examples are scaled accordingly. The examples from all classes are scaled in this way. This allows the scores of examples from different classes to be compared. An example with a score of 1 is the most representative of its class; the example with the lowest score is the least representative of its class.

CN2-U has a user supplied threshold which controls how many of the poorest scoring examples should be removed. CN2-U sorts the examples according to how well each example scored, with the most representative examples at the top. The threshold has a value in the range {0:1} where a value of 0 would treat every example as unrepresentative and a value of 1 would treat evey example as representative (using the threshold set to 1 will result in CN2-U behaving as CN2).

Table 6.5 shows the d-scores obtained by CN2-U and from this table it can be seen that examples 2 and 7 would also appear at the bottom of the sorted examples.

These examples would then be considered the least representative in the training set. If the threshold was set to 0.7, both these misclassified examples would be identified as unrepresentative by CN2-U.

Table 6.6 shows the classification accuracies and ruleset complexities

example No	class	Attribute				d-score(CN2-U)
		covering	legs	wings	colour	
1	bunny	fur	4	no	grey	1.0
2	bunny	feathers	2	yes	black	0.536
3	bunny	fur	4	no	brown	1.0
4	owl	feathers	2	yes	grey	0.9
5	owl	feathers	4	yes	brown	0.9
6	owl	feathers	2	yes	white	1.0
7	owl	fur	4	no	white	0.6

Table 6.5: D-scores obtained by CN2-U for the owls and bunnies

found by comparing CN2-D and CN2-U, using the five learning problems described above. More particularly, Table 6.6 shows the classification accuracies of CN2-U and CN2-D for similar ruleset complexities. This method of evaluation is used because of the different methods employed by CN2-U and CN2-D to identify unrepresentative/difficult examples. CN2-U uses a threshold, by which the number of examples deemed unrepresentative can be varied. In contrast, CN2-D operates a 'cut-off' strategy which cannot be varied. CN2-U was tested by altering the threshold value in range 0.5 to 1, which gives a variety of rulesets, the complexity of which decreases as the threshold value decreases. The best method of evaluating the performance of CN2-U and CN2-D is thus to compare the classification accuaracy of CN2-U and CN2-D when the threshold value for CN2-U produces a rulest of similar

complexity to that induced by CN2-D. In theory CN2-U can induce rules which have only one condition, however, these rules would have a very low classification accuracy, thus comparing classification accuracies for rulesets of a similar complexity is a more meaningful comparison. In Table 6.5 the learning problems names have been abbreviated as follows: CL (Contact Lenses); MS (Mushrooms); M1 (Monks 1); M2 (Monks 2); M3 (Monks 3).

Problem	Ruleset Complexity		Classification Accuracy	
	CN2-D	CN2-U	CN2-D	CN2-U
MS	2.67	2.33	45	9.09
CL	1.67	2.33	70	45.83
M1	12	12	80	86.8
M2	31	38	58	57.98
M3	7	14	97.2	55.94

Table 6.6: Classification accuracies and auleset Complexities for CN2-D and CN2-U for the contact lenses, mushrooms and Monks Problems

From Table 6.6 it can be seen that the rulesets obtained by CN2-D have a higher classification accuracy than CN2-U for all problems other than Monks 2. These results show that CN2-D, the strategy where the d-score is used to identify difficult examples which are then removed, is a better way of reducing ruleset complexity than CN2-U i.e the less complex rulesets induced by CN2-D have a higher classification accuracy than the comparable rulesets induced by CN2-U. This is because the examples identified as difficult by CN2-D are

both unrepresentative of their class *and* more representative of another class. The results of this experiment show that CN2-D employs a more effective complexity reduction strategy that CN2-U; thus the performance of CN2-D will be further evaluated. CN2-U will not be considered any further in this thesis.

6.3.2 Experiment 2: A Multi-Strategy System, CN2-D with Nearest Neighbour

Figure 6.1 illustrates the components of CN2-D+NN: a multistrategy learning system developed by the author in which the simple ruleset induced by CN2-D is combined with a simple nearest neighbour algorithm. CN2-D+NN has a learning phase and a test phase. The learning phase uses the d-scores to split the training set into an easy subset and a difficult subset. As for CN2-D, the CN2 algorithm is used to induce a ruleset from the easy subset. The difficult subset represents a set of 'cases' to which a test set can be compared. The test phase first determines whether the classification of a test example should be provided by the ruleset or by the difficult cases. This is achieved by obtaining a set of similarity measures showing how similar a test example is to each of the difficult cases. The similarity measure then counts the number of shared attributes. The similarity scores are compared to a threshold to determine whether the classification should be given by the difficult cases or by the ruleset. If the similarity score for any comparison with a difficult case is below this threshold, then the test case is given by the difficult case (or cases) to which the test example is most similar. If no comparison is lower than the threshold, then classification is provided by

the ruleset. If difficult cases from more than one class are equally similar to the test example, then the test example is given the same classification as the most frequent class found in these equally similar cases. The threshold can be changed so that the different components of the system can be given different proportions of the test set.

Figure 6.2 shows the classification accuracies obtained by applying CN2-D+NN to the five learning problems used in Experiment 1. It also shows how classification accuracy varies as the threshold varies.

It can be seen that for all problems other than Monks 2 the highest classification accuracy was obtained by setting this threshold to 0. With the threshold set to 0 CN2-D+NN behaves exactly as CN2-D, i.e. no test examples are classified by the nearest neighbour component. The highest classification accuracy obtained for Monks 2 is for a threshold of 1, where a minimum number of test examples are classified by the nearest neighbour component. It can be seen from this result that combining the simple ruleset induced by CN2-D with a nearest neighbour component to classify difficult examples performs more poorly than CN2-D, where the difficult examples are ignored. CN2-D+NN will be given no more consideration in this thesis.

6.3.3 Experiment 3: Does Removing Difficult Examples Give Better Performance Than Removing Easy Examples ?

Experiment 3 has been designed to test whether removing those examples identified as difficult by CN2-D results in better performance than removing a similar number of examples identified as 'easy' by CN2-D. Figures 6.3 and

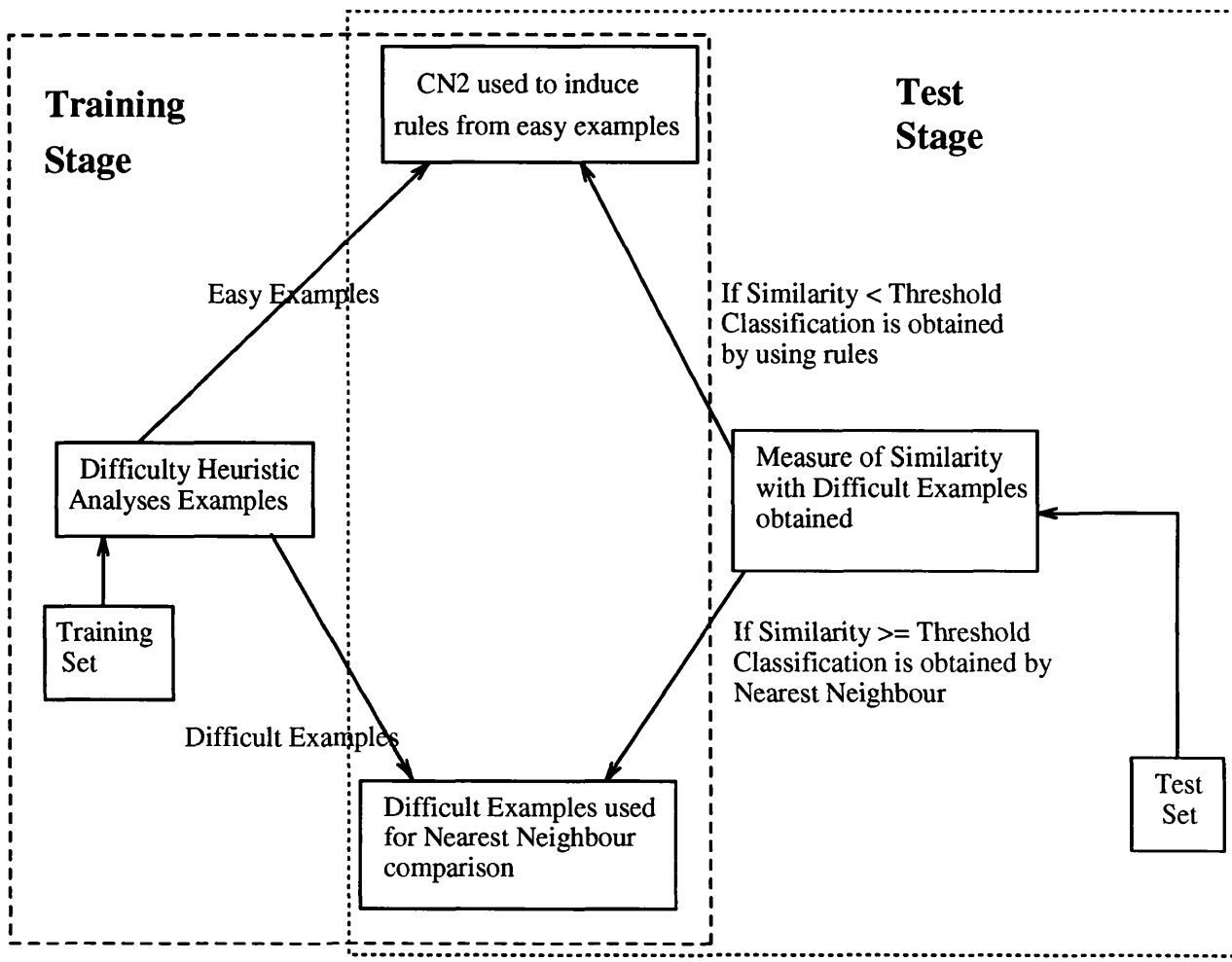

Figure 6.1: CN2-D+NN: A multi-strategy system incorporating Induction and Nearest Neighbour

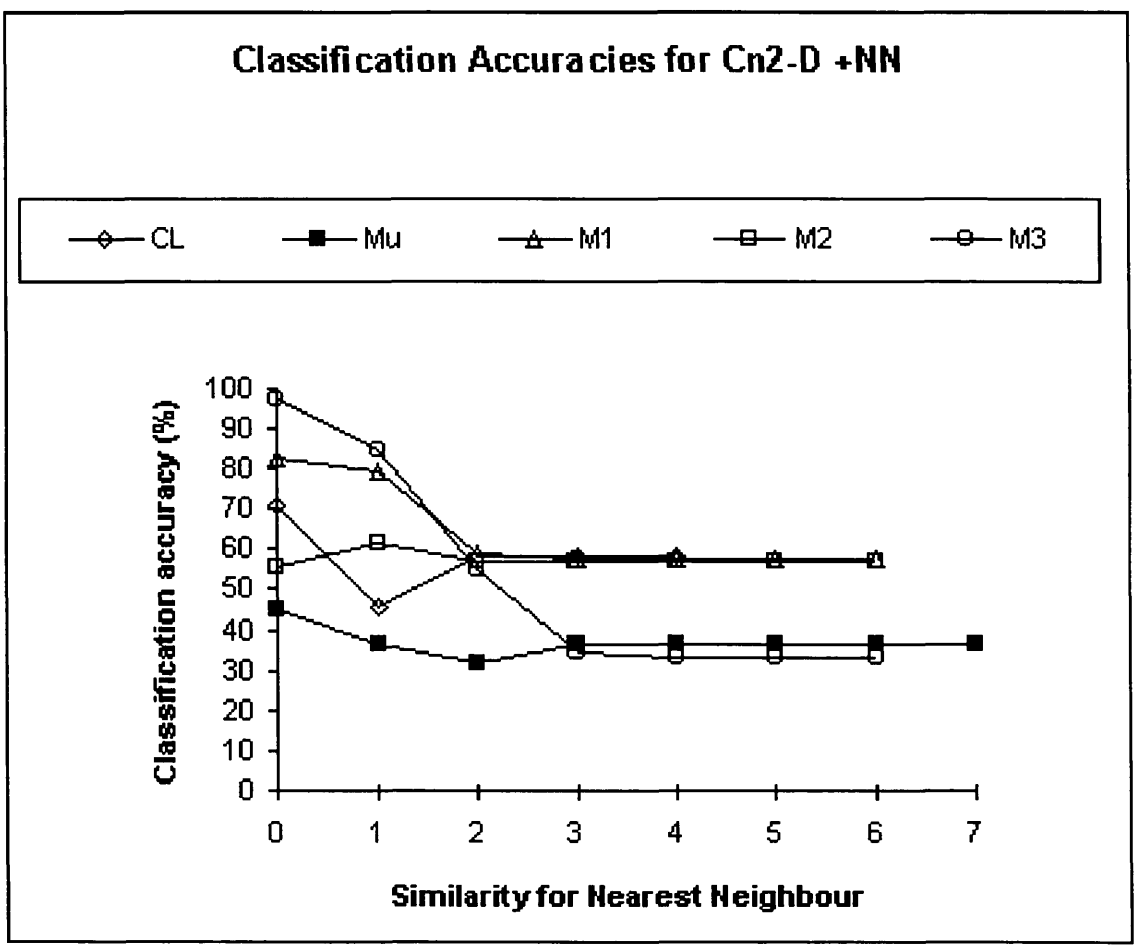

Figure 6.2: Classification accuracy obtained by CN2-D+NN

6.4 illustrate the classification accuracy and ruleset complexity results for this experiment. For purposes of this experiment, the strategy of removing easy examples is termed *CN2-E*. To evaluate more closely the effects of removing easy examples from the training set, any difficult examples identified by the d-score were retained. CN2-E counts the number of difficult examples found and removes the same number of easy examples from the training set. A random procedure was used by CN2-E to identify those easy examples which should be removed.

Figures 6.3 and 6.4 show that for the Mushrooms, Contact Lenses and Monks 3 problems removing the difficult examples has resulted in rulesets which are substantially less complex and also more accurate than those induced by the strategy of removing easy examples.

For monks 1, CN2-E induces a ruleset of similar complexity to CN2-D, however this ruleset is some 5% less accurate in classifying the test set. Monks 2 is the only problem for which CN2-E induces a more accurate ruleset. However, this ruleset is nearly three times as complex as that induced by CN2-D. As discussed more fully in the next section, the characteristics exhibited by Monks 2 suggest that the strategy of identifying and removing difficult examples performs poorly where a learning problem displays representational complexity. The results for all of the other learning problems suggest that identifying and removing difficult examples, and 'ignoring' the information contained within them is the best of the complexity reduction strategies described in this thesis. All further experiments will thus consider CN2-D only.

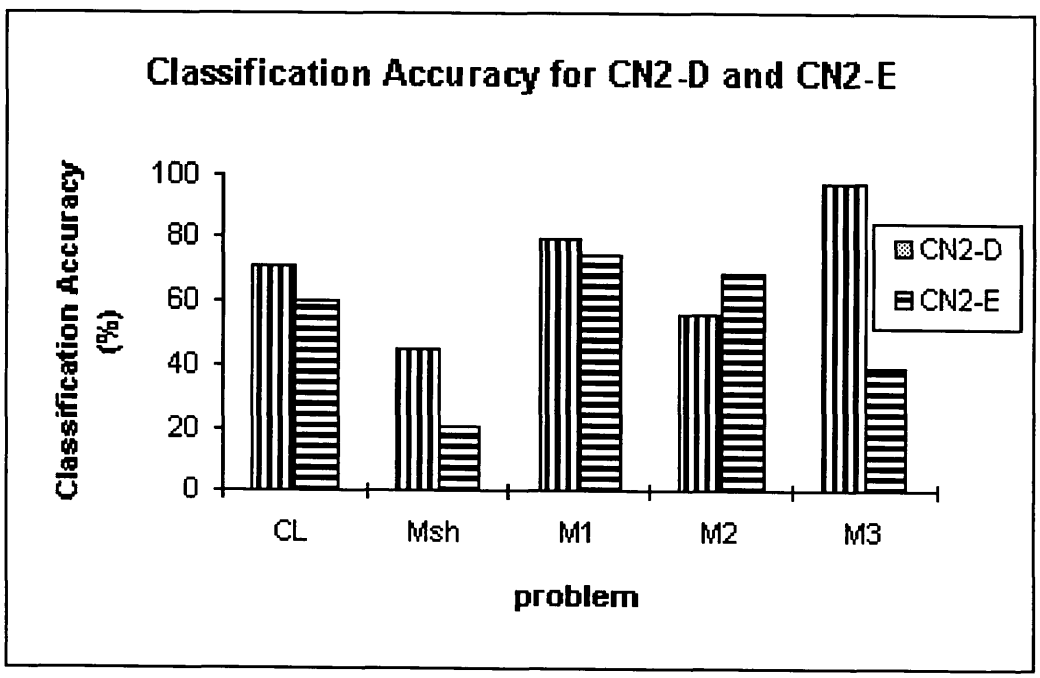

Figure 6.3: Classification accuracies for CN2-D and CN2-E

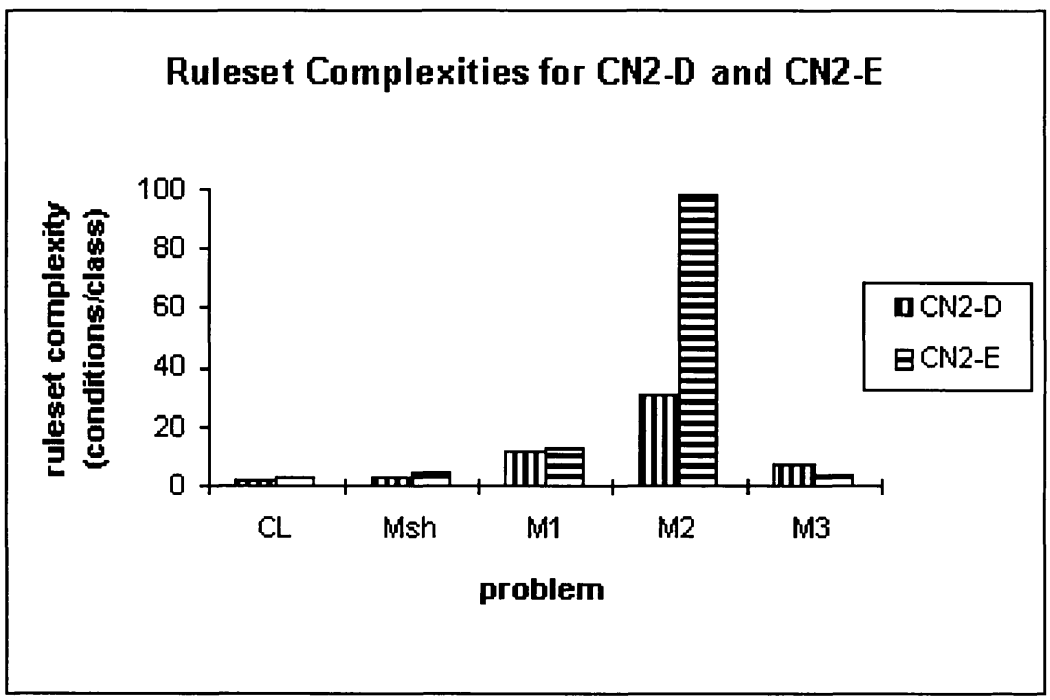

Figure 6.4: Ruleset complexities for CN2-D and CN2-E

6.4 A Preliminary Comparison of CN2 and CN2-D

For this experiment, results pertaining to the DEFAULT rule have been included. The DEFAULT rule gives the classification accuracy obtained by merely assuming that all test examples belong to the most frequently occuring class found in the training examples. This has been included to investigate how much better the induction strategies are than this very simple method. The number of difficult examples identified by CN2-D in each of the problems on which CN2 and CN2-D were compared is shown in Table 6.7

Problem	No of Difficult Examples	No of Examples	% of Difficult Examples
Contact Lenses	3	12	25.00
Mushroom	9	23	39.13
Monks 1	29	124	23.39
Monks 2	71	169	42.26
Monks 3	9	122	7.37

Table 6.7: Number of difficult examples found in the Contact Lenses, Mushrooms and Monks Problems

From this table it can be seen Monks 2 and the Mushrooms contain a significantly higher proportion of examples identified as difficult by CN2-D than the other problems. Monks 3 contains the least number of difficult examples. Monks 1 and the contact lens problems are intermediate in difficulty, containing 25% and 23.39% difficult examples respectively.

6.4.1 Ruleset Complexity

The complexity of the rules induced by both CN2 and CN2-D is illustrated in table 6.8 and figure 6.5. Table 6.8 shows the average number of conditions per class for the different problems.

Problem	Ruleset Complexity	
	CN2	CN2-D
Contact Lenses	3	1.67
Mushrooms	8.67	2.67
Monks 1	32.5	12.0
Monks 2	111.0	31.0
Monks 3	32.0	7.0

Table 6.8: Rule complexities for algorithms for Contact Lens, Mushrooms, and Monks problems (measured as average number of conditions per class)

These results show that Monks 2 produces the most complex ruleset, with 111 conditions per class. Corresponding, Monks 2 shows the greatest reduction in rule complexity. Monks 3 also shows a marked decrease in rule complexity, from 32 to 7 conditions per class. From these results it can be seen that the rulesets produced from training sets with the difficult examples removed have often more than 50% fewer conditions per class and more than 50% fewer rules than rulesets produced by learning from the entire data sets. However, as table 6.9 illustrates, there is no simple correlation between the amount of difficulty in a dataset and the resultant reduction in ruleset

complexity.

Problem	%Difficulty	Reduction in Complexity (CN2 - CN2-D)
Contact Lenses	25.00	1.33
Mushrooms	39.13	6
Monks 1	23.39	20.5
Monks 2	42.26	80
Monks 3	7.37	25

Table 6.9: The reduction in complexity and % difficulty of the learning problems used

Monks 2 shows the greatest reduction in complexity and correspondingly had the highest difficulty. However, the Contact Lenses had a high difficulty of 25% but the reduction in complexity was only 1.33 conditions per class. In contrast Monks 3 had only 7.37% difficulty but ruleset complexity was reduced from 32 to 7 conditions per class.

Figure 6.5 shows that removing difficult examples from the learning process also shifts the distribution of rules towards those rules with fewer conditions. This shift in distribution is most apparent in Monks 3 where the percentage of rules with 1 or 2 conditions increases from 39.1% to 100% as the difficult examples are removed. It shows how the number of rules is more than halved in all problems except the contact lenses problem where the reduction is 40%. The Mushrooms and Monks 3 show the greatest decrease in ruleset size: from 14 to 5 rules for the Mushrooms and 23 to 8 for Monks 3.

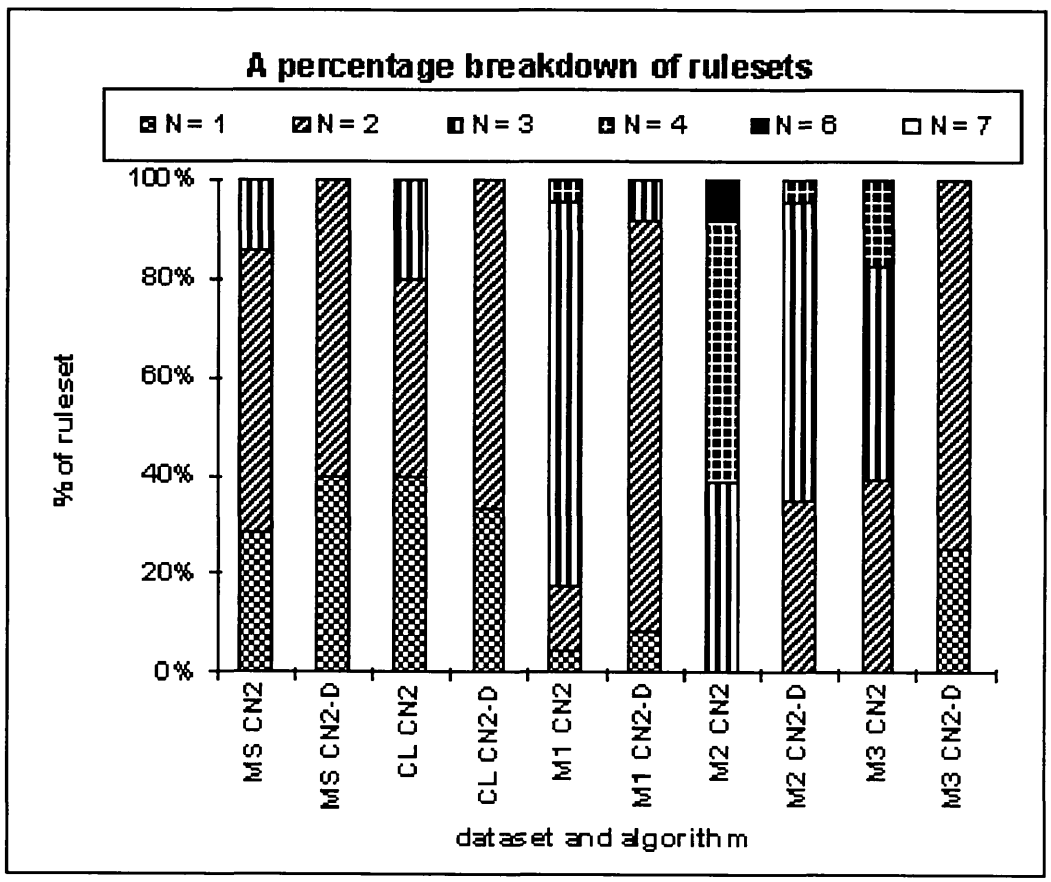

Figure 6.5: A percentage breakdown of rulesets according to the number of conditions (number of conditions given by "N")

6.4.2 Classification Accuracy

The classification accuracy of CN2 and CN2-D on the different datasets is presented in figure 6.6. These results show that for Monks 1 and 2, CN2-D has classification accuracy of 12% and 13% less than CN2. The classification accuracy of CN2-D is higher than CN2 for the Mushrooms, Contact Lenses and Monks 3 problems. There is an increase of 15% for the Contact Lenses, 8% for the Mushrooms and 10% for Monks 3. If these differences are expressed as a percentage of the classification accuracy of CN2, then CN2-D performs as follows. For the Mushrooms, Contact Lenses and Monks 3 CN2-D performs 21.4%, 17.58% and 8.9%, better than CN2, respectively. For Monks 1 and 2, CN2-D performs 13.2% and 19.1%, worse than CN2, respectively.

6.4.3 Discussion and Conclusion

The above results have been obtained by comparing the performance of CN2 and CN2-D on data sets which have been created to illustrate the different types of difficulty common to real world machine learning problems. From the rule complexity results it can be seen that CN2-D, a learning system that identifies those elements of a training set that are easy to learn from and then applies CN2 only to those examples, always produces rules that were substantially less complex than those produced by applying CN2 to the whole training set. In contrast to the other pruning methods described in section 2 that use statistical methods for terminating the rule growing process before overspecialised rules are produced, CN2-D avoids learning

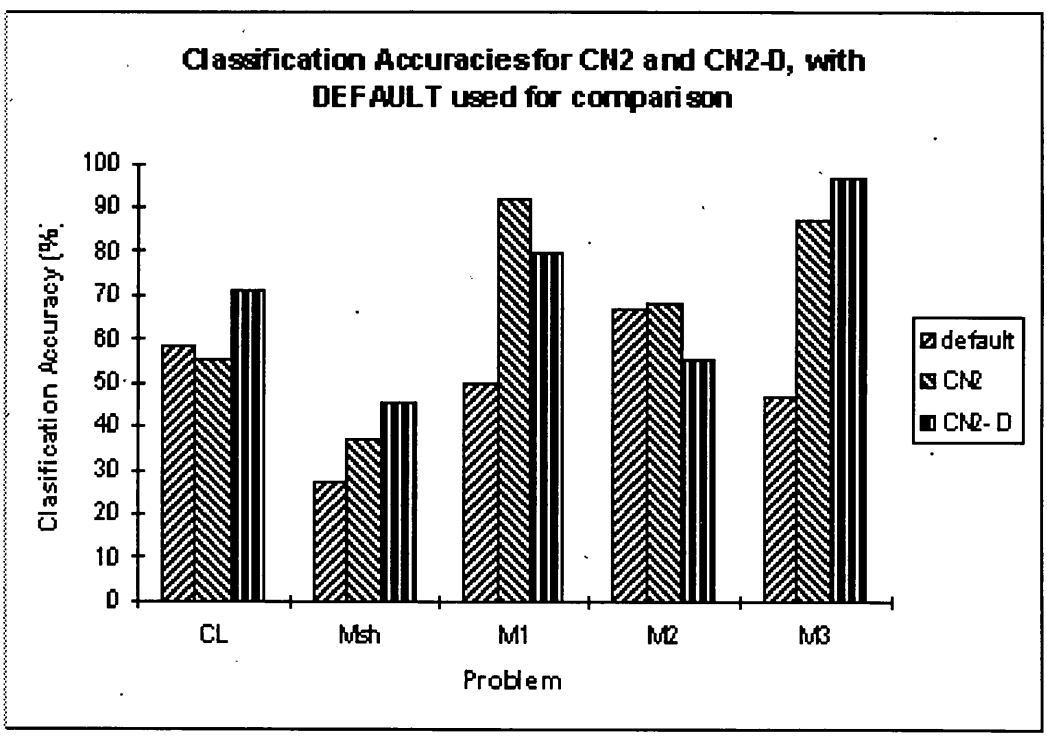

Figure 6.6: Classification accuracies for the learning problems

from those examples in the training set, and thus avoids the induction of over-specialised rules. Thus, when difficult situations are encountered, that would usually lead to the induction of a large number of small disjuncts, this system avoids producing many small disjuncts by not learning from difficult examples.

However, although CN2-D is successful at reducing the complexity of rulesets, as the classification accuracy results show, the relationship between the reduction in ruleset complexity and classification accuracy is not simple, CN2-D has a higher classification accuracy than CN2 for the Contact Lenses, Mushrooms and Monks 3 problems but a lower classification accuracy for Monks 1 and 2. The behaviour of the algorithms on each learning problem will be discussed in detail below.

The Mushrooms problem is difficult because it contains duplicate examples which belong to different classes. The number of attributes used is not sufficient to properly distinguish different members of the population. This situation poses problems for CN2, and also for the pruning methods which allow for a number of incorrect training examples to be covered by a rule. Producing rules with an allowable error will increase inaccuracy if more duplicates with the incorrect classification are found in the test set. However, using CN2-D to identify difficult examples can give induction a better chance of choosing the correct classification for unseen duplicates by learning only from that duplicate which is more like other members of its class.

Monks 3 is a simple problem in which difficulty is introduced in the form of misclassifications. The low percentage of difficult examples found in this problem is not an accurate measure of how difficult it is to learn from, as is

shown by the increased complexity of the ruleset required by CN2 to describe the problem. CN2 performs poorly in this case because it produces inaccurate small disjuncts in an attempt to cover the misclassifications found in the training set. The misclassifications are identified by CN2-D and are removed. The induction of simpler rules from the remaining examples allows CN2-D to classify the test set with a high classification accuracy (97%).

The Monks 2 problem is a learning situation in which CN2-D performs more poorly than CN2. This is because the difficulty in this problem is representational in nature. The attribute value pair representation language is inadequate for representing this problem. The concept described by Monks 2 is a parity problem: a training example belongs to the positive class if any *two* attributes have their *first* value (Thrun et al 1991). All other examples belong to the negative class. This is a complicated class distribution which cannot easily be described using rules containing attribute value pairs, and is the reason for the large number of rules and the high complexity of the ruleset produced by CN2. Small disjuncts in this case are necessary for describing this problem; removing the examples which cause them has merely reduced the ability of CN2 to represent the problem to its best ability. It has been shown that induction algorithms that change the representational space by constructive induction perform well on this problem (Thrun et al 1991). It should be noted that the CN2 induction algorithm also only marginally outperforms the DEFAULT score, and thus cannot be considered successful at inducing rulesets from this problem.

The Contact Lenses and Monks 1 problems are comparable because they are both complete populations with no misclassifications, no duplicates and

are easily represented in the attribute-value based language used to form the rules. In this situation some of the class distribution may be better described by rules which have been induced to cover the difficult examples, which would be the case in Monks 1; or, as in the Contact Lenses problem, the difficult examples produce rules which inaccurately reflect the class distribution of the problem. There seems to be no simple way to predict which strategy is best for inducing rules from simple datasets with no noise.

6.5 C4.5-D Can Identifying and Removing Difficult Examples Give Simpler Decision Trees?

This experiment was undertaken to ascertain whether identifying and removing difficult examples from training sets which were then presented to C4.5 resulted in a similar performance, where in most cases the marked reduction in ruleset complexity is not matched by a corresponding reduction in classification accuracy. Ruleset complexity for C4.5 is scored in exactly the same way as CN2, using the rulesets which result from treating each branch of the decision tree as a rule. The system built from applying the strategy used by CN2-D to identify difficult examples, but using C4.5 to induce decision trees from easy examples, is known as C4.5-D.

Figures 6.7 and 6.8 show the classification accuracies and ruleset complexities obtained by applying C4.5 and C4.5-D to the five learning problems used in the other comparisons described in this chapter.

These figures show that the strategy of identifying and removing difficult

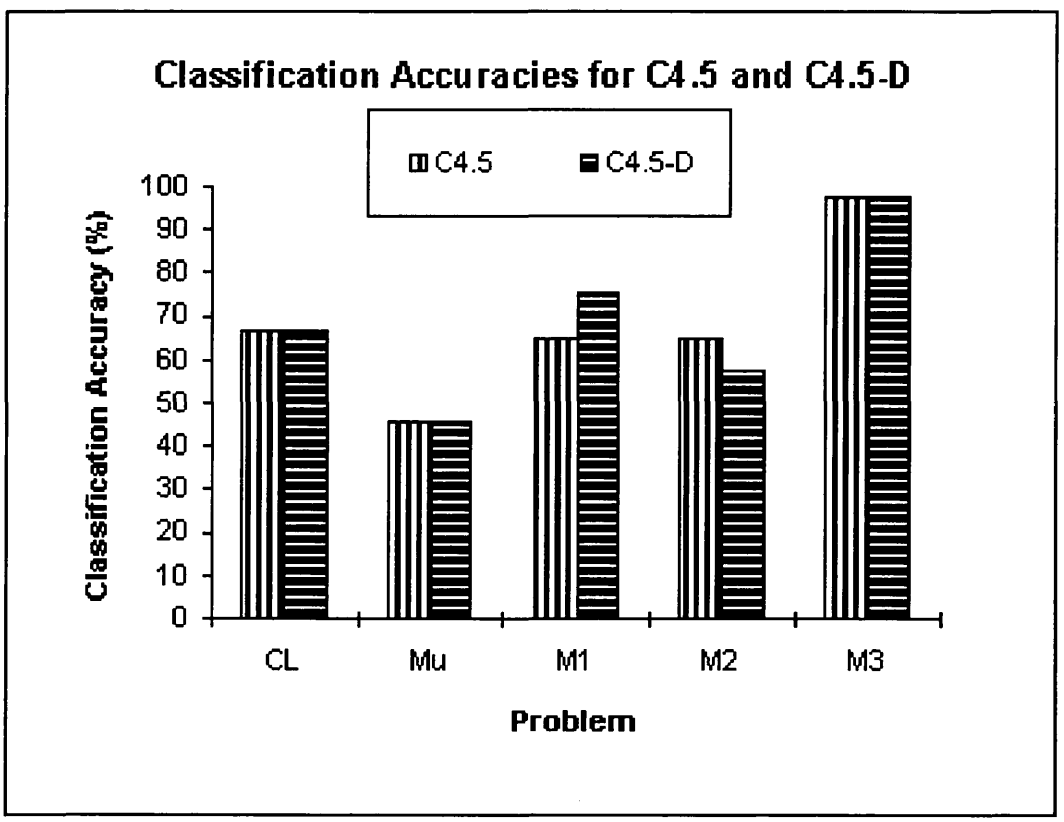

Figure 6.7: Classification accuracies for C4.5 and C4.5-D

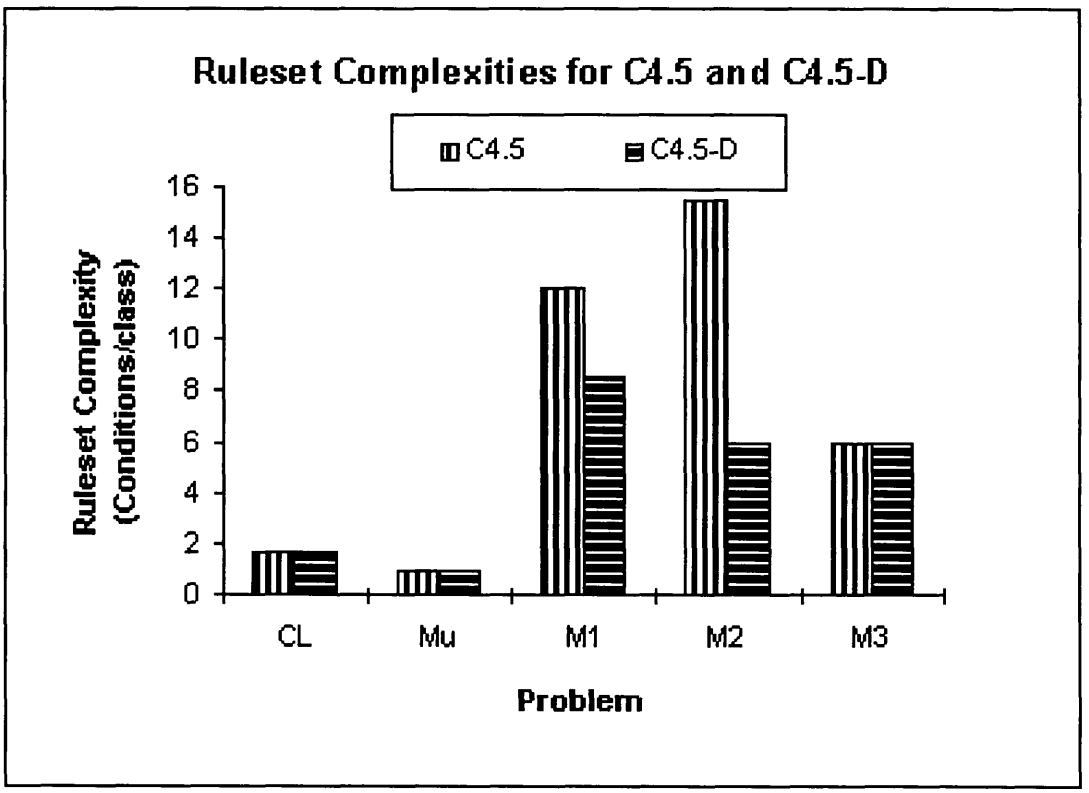

Figure 6.8: Ruleset complexities for C4.5 and C4.5-D

training examples has no effect on the performance of C4.5 for the Contact Lenses, Mushrooms and Monks 3 problems. The strategy produces a reduction in complexity for Monks 1 and Monks 2, with Monks 2 showing a larger reduction. However, this reduction in complexity brings about an increase in classification accuracy for the Monks 1 problem, and a reduction in classification accuracy for the Monks 2 problem. The reduction in classification accuracy for Monks 2 may be explained in the same way as the similar results for CN2-D i.e that the difficult examples removed from the training set represent genuine complexity and the easy training set does not contain a sufficient number of examples to correctly represent the class distributions found in the problem. The results from the Contact lenses, Mushrooms and Monks 3 problems arise from the difference in learning strategy between rule based and decision tree induction. In the case of decision tree induction removing a number of examples has had no effect on the selection of which attribute to place in the decision tree at any particular stage in its growth. In contrast, removing examples has a large effect on how attribute values are scored by the preference criterion. It should also be mentioned that the post-pruning methods employed by C4.5 provide an effective way of reducing ruleset complexity with little cost to classification accuracy.

6.6 Conclusion

From these experiments it can be seen that the best strategy for reducing ruleset complexity without a corresponding reduction in classification accuracy is that employed by CN2-D: i.e identifying and removing difficult examples

and using CN2 to induce a simple ruleset from the remaining easy examples. Further experiments which attempt to utilise the difficult examples were also shown to have an adverse effect on classification accuracy. Using the d-score to preprocess training sets for C4.5 (C4.5-D) also had a poorer result than that found by applying the d-score to CN2.

From the comparison of CN2-D and CN2, the result for Monks 3 revealed that for this problem the simpler ruleset induced by CN2-D had a higher classification accuracy. Monks 3 was the only dataset which contained misclassifications. This result suggests that the strategy of inducing simple rulesets from easy training examples works well on noisy data. The next chapter, therefore, presents a series of experiments in which CN2 and CN2-D are used to induce rules from noisy datasets. To obtain a wider picture of how this strategy behaves, learning problems that display a range of characteristics have been used for evaluation purposes. Results will also be obtained for a range of noise levels so that the behaviour of the algorithms as noise increases can be observed.

Chapter 7

Inducing Rulesets from Training Sets Containing Misclassifications

The last chapter presented some preliminary discussion on inducing succinct rules by firstly identifying the best strategy for complexity reduction and secondly, comparing the performance of CN2 and CN2-D on a number of small problems. To further evaluate the performance of these systems with noisy data, experiments of a more detailed nature were undertaken. A number of contrasting datasets were obtained, so that the performance of the two systems could be measured in different learning situations. Initially the training sets contained no noise. However noise was added by altering the classification of a pre-specified number of examples. The performance of the two systems can then be measured for a range of noise levels. The following sections discuss the datasets and the nature of any complexity found within them, the experimental method used generate the training sets and noise levels, and the evaluation criteria used to measure performance.

7.1 Experimental Conditions and Learning Problems Used

Six learning Problems were selected on which the performance of CN2 and CN2-D could be compared. All of the examples comprising the problems had to be described in terms of attribute value pairs with the value sets of each attribute restricted to categorical attributes. This is because the present implementation of the d-score metric is restricted to categorical attributes. A proposed extension to include ordered and continuous attributes is discussed in chapter 8. The six problems include the three Monks problems described in the previous chapter, two datasets obtained from the UCI Machine Learning Repository (Murphy and Aha 1994) and a Thyroid dataset avaliable at the University of Abertay, Dundee (UAD) (File et al 1994). The misclassifications introduced into the Monks 3 dataset by Thrun et al (1991) were corrected before any misclassifications were added, so that a noise-free result could also be obtained. The Thyroid dataset corresponds to a complete real world problem which is described by 7 discretely-valued attributes, and classified by a classification attribute with three possible values. These data were collected by previous researchers in the Informatics Department at UAD.

The two datasets obtained from UCI were the Tic-Tac-Toe (Noughts and Crosses) dataset and the Hayes Roth dataset. Tic-Tac-Toe is the complete population of the 958 possible Tic-Tac-Toe end game configurations. Tic-Tac-Toe is a two player game with player "x" and player "o". For this dataset it is assumed that player x moves first. The configurations are described using 9 attributes: each corresponding to one of the 9 board positions,

e.g top left, bottom right, middle left etc. Classification of a configuration is achieved by assigning a win for player x to the positive class and all other outcomes to the negative class. The game is won by player x when xs occupy a "three in a row" position i.e one single row, one single column or one of the diagonals. Previous results using Tic-Tac-Toe shows that rule-based learning algorithms such as CN2 perform well (98.2%) (Aha 1991) and decision tree algorithms such as ID3 perform poorly. However this dataset is often used to test constructive induction systems because the crucial information required to classify the examples is carried by the relationship between the squares on the board rather than which square contains "x" or "o". This quality is shared by the chess end game problem described in the constructive induction section 5.4. The additional complexity of an attribute value based representation is clearly demonstrated: many rules are required to classify the large numbers of examples belonging to the target classification because each winning configuration is considerably different from the others. However a predicate logic learning system such as FOIL can take advantage of three place predicates such as:

diagonal(X,Y,Z)

which can more concisely represent the relationships present in the game. Constructive induction would aim to augment the original propositional attribute space by building new attributes which combine the original attributes e.g.

[top_left_and_middle_left] = "x"

Problem	Problem Characteristics			
	size	real/artificial	% of difficult examples	type of complexity
Monks-3	432	artificial	7.37	normal
Thyroid	840	real	14.5	normal
Hayes-Roth	132	real	18.2	normal
Monks-1	432	artificial	23.39	normal
Tic-Tac-Toe	958	real	30.4	representational
Monks-2	432	artificial	42.26	representational

Table 7.1: Learning problems used and problem characteristics

The Tic-Tac-Toe learning problem thus has a high complexity due to the inadequacy of the attribute-value based knowledge representation.

Hayes-Roth is a small dataset containing 132 members. It has 5 attributes concerning human behaviour. It has been used to test instance-based learning systems. It is a simple dataset with little complexity. A summary of the datasets and their complexities is given in table 7.1. In this table the term "normal" complexity is used to describe learning problems in which representational complexity is not found. These problems will also have a number of difficult examples, but this difficulty arises from the distribution of the examples rather than the inadequacy of the attribute-value pair representation language. It can be seen from table 7.1 that those learning problems showing representational complexity have a higher proportion of difficult examples.

7.2 Experimental Conditions

The datasets for each learning problem were presented to CN2 and CN2-D both to induce the rulesets (training) and to test the performance of the rulesets (test). To overcome any idiosyncrasies within training sets, five noise free training sets were randomly chosen from each learning population. Each training set contained 25% of the learning population. In each case the test set included the entire population of the learning problem. Results were obtained for the initial, noise free, experiments before each of the five training sets for each problem were altered to simulate noisy training data. Noisy training examples were created by changing the classification attribute of a predetermined number of training examples. This number of training examples corresponded to the percentage of noise desired: it was decided to record the performance of CN2 and CN2-D for noise levels from 5% to 50% with a 5% increment: i.e 5% noise, 10% noise 15% noise ... 50% noise. Again to offset any idiosyncrasies in the choice of which training examples to change, 10 new training sets were produced for each noise level, with the examples to be altered chosen at random. The classification attribute of an example was altered systematically. For the learning problems where the classification attribute had two possible values (Monks 1 to 3, Tic-Tac-Toe) the noise-free values were replaced by the other value (i.e those examples from class 1 now belonged to class 2 and vice versa). Classifications for the learning problems with 3 values (Thyroid and Hayes-Roth) for the classification attribute were altered as follows:

examples from class 1 now belonged to class 2

examples from class 2 now belonged to class 3

examples from class 3 now belonged to class 1

As in the previous chapter classification accuracy and ruleset complexity were used as the performance measures to evaluate whether any observed decrease in ruleset complexity brought about a corresponding decrease in classification accuracy.

7.3 Evaluation of Results

7.3.1 Ruleset Complexity and Classification Accuracy

Ruleset Complexity

Figure 7.1 displays the differences in ruleset complexity (%) between CN2-D and CN2 for the six learning problems used in the experiment. These differences are given for each 5% increment in the range 5% to 50%. Each difference figure was obtained by subtracting the value of CN2 from the value of CN2-D. The results used to obtain the differences were the mean values for each algorithm, averaged over all five training sets for each problem. A mean value for each noise level in each training set was obtained by averaging the 10 results for each algorithm for that noise level. These results are shown in figure 7.2. CN2-D always produced rulesets which were less complex than CN2. Four of the problems (Tic Tac Toe, Thyroid, Monks 3 and Monks 1) show very steep initial curves which decrease in gradient after 25% noise.

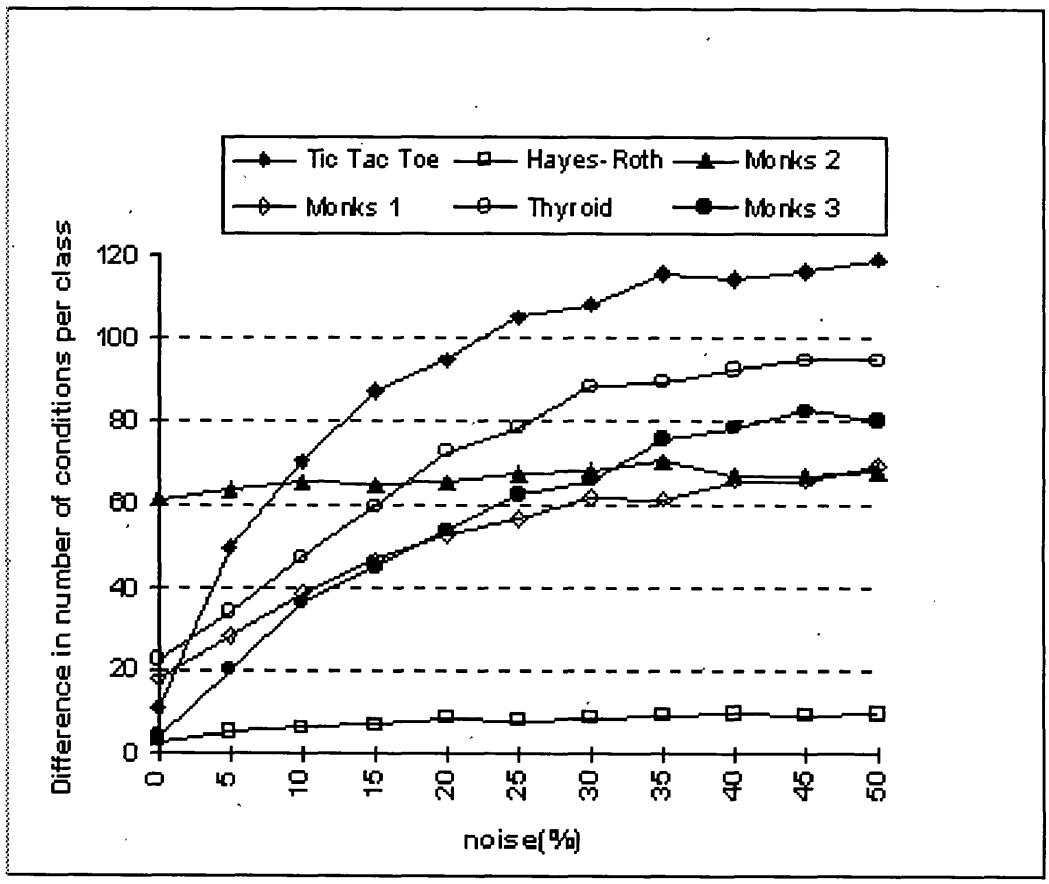

Figure 7.1: Differences in ruleset complexity

The Tic Tac Toe increase is particularly large, with CN2 producing rulesets with 10 conditions per class more than CN2-D at 0% noise and 120 more conditions per class at 50% noise. The other two problems have much flatter curves, however these are at very different values. For the Monks 2 problem CN2 produces rules which are 60 conditions per class more complex than CN2-D at 0% noise: this only increases to 68 conditions per class at 50% noise: thus CN2 always produces rules which are much more complex than CN2-D. The curve for Hayes-Roth shows that there is much less difference in ruleset complexity for this problem: CN2 is more complex by only 4 conditions per class at 0% noise, this increases slightly to 10 conditions per class at 50% noise

Classification Accuracy

Figure 7.3 shows the difference in ruleset complexity (conditions per class) for the six learning problems. The calculation of these differences was performed in a similar way to the calculations for figure 7.1. Figure 7.3 contains a negative region as well as a positive region. The positive region of figure 7.3 represents situations where CN2-D has a higher classification accuracy than CN2 and the negative region represents the opposite situation; where CN2 has a higher classification accuracy than CN2-D. Figure 7.4 contains a set of line graphs of the classification accuracy results for each problem.

The curves representing the classification accuracies (figure 7.3) for the Monks 3 and Thyroid problems are almost entirely in the positive region of the graph: only in the noise free case for Monks 3 is there a small negative

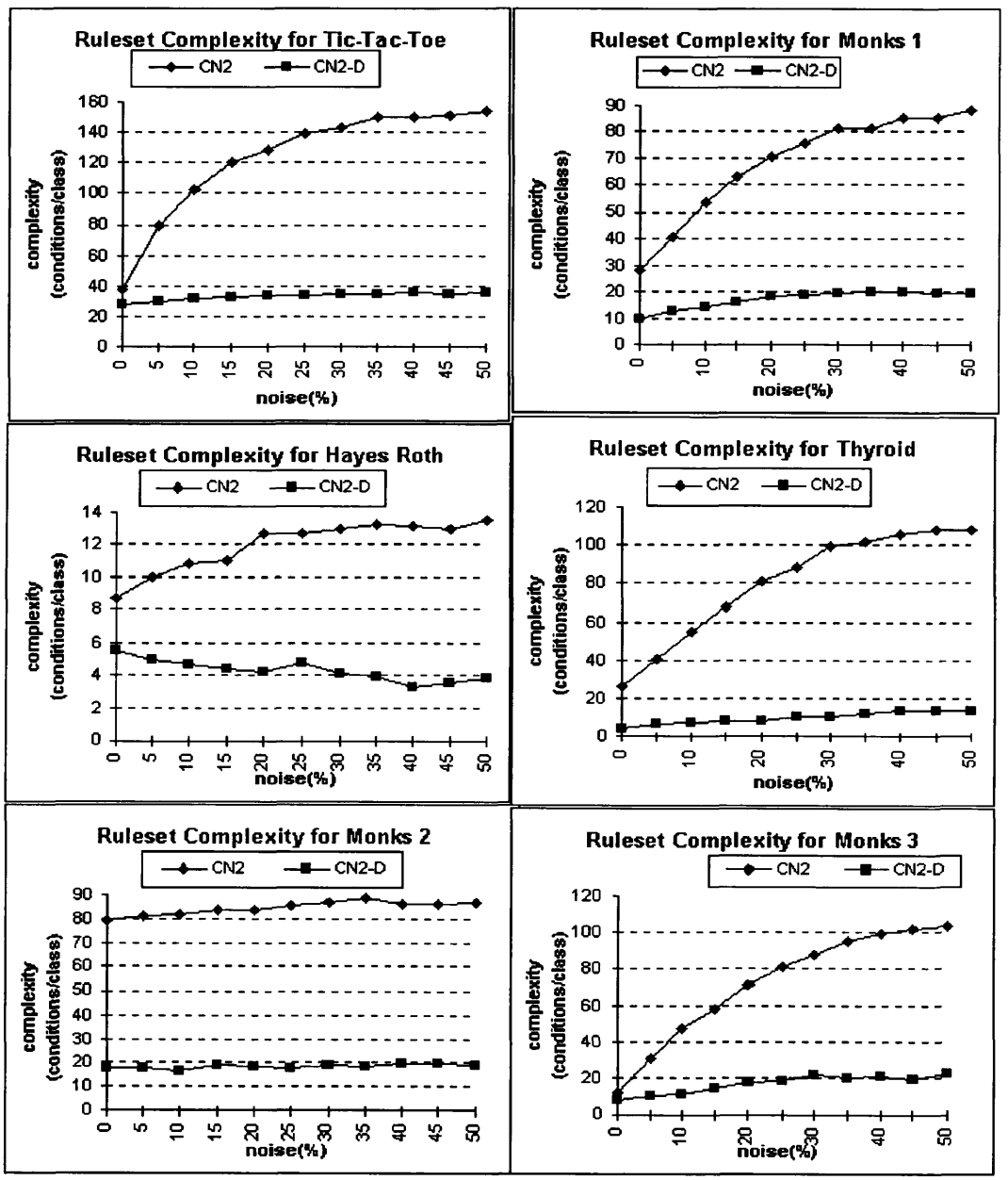

Figure 7.2: Ruleset complexity

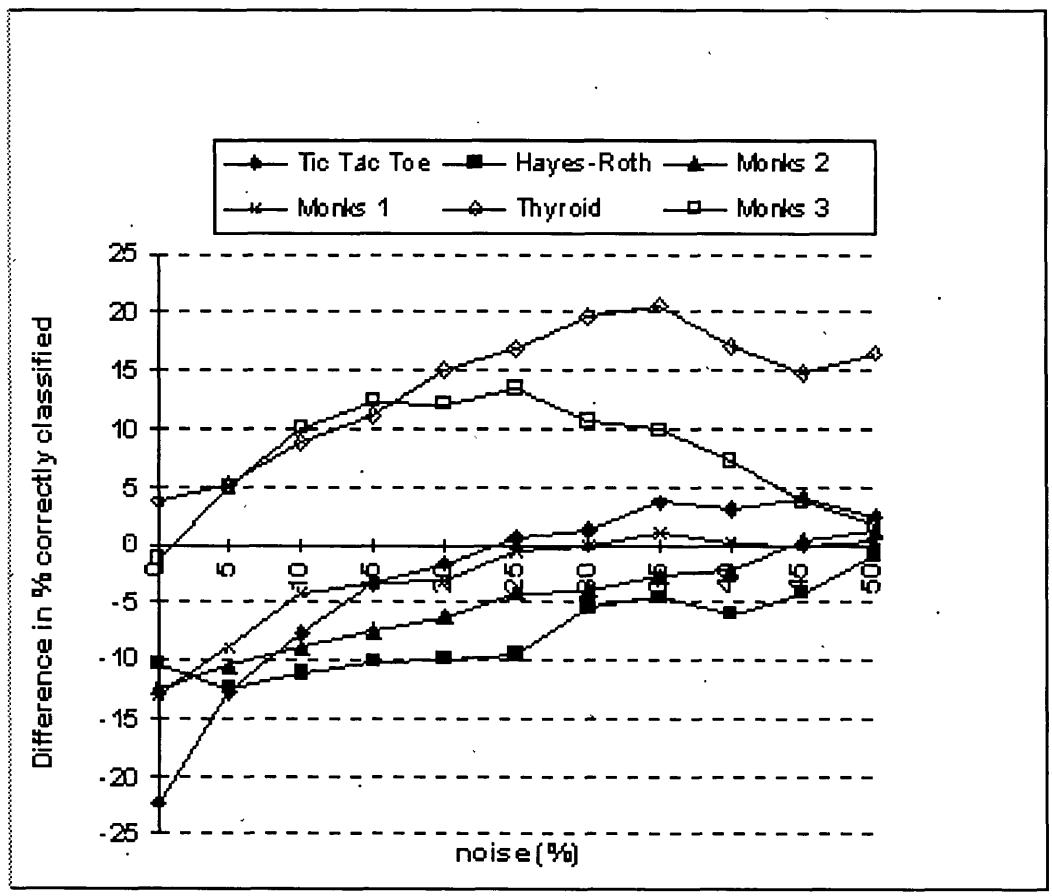

Figure 7.3: Differences in classification accuracy

result. Thus for these two problems CN2-D has a higher classification accuracy than CN2. There is a steady increase in the height of the Monks 3 curve from 0 to 15% noise, where CN2-D has a higher classification accuracy than CN2 by nearly 15%; and a steady decrease from 25 to 50% noise, where the difference has decreased to 2%. The Thyroid curve also shows a steady increase, from 0 to 35% noise, where CN2-D has a higher classification accuracy than CN2 by 20%. From this point the height declines to value of 16%

All of the other curves start in the negative section of figure 7.3 and finish either slightly above or below zero, indicating that, for these problems, CN2 starts out having a higher classification accuracy than CN2-D for these problems. The Tic Tac toe curve starts at a value of -22%, but rises steeply to a value of 4% at 35% noise, thus CN2 has a higher classification accuracy for the less noisy portion of the Tic Tac Toe experiment, but from 25% noise onwards CN2-D has a higher classification accuracy than CN2. A similar situation is displayed by Monks 1: CN2 has a higher classification accuracy for the lower levels of noise, but from 25% noise onwards the algorithms show a similar classification accuracy. Hayes-Roth is the only problem for which CN2-D has a higher classification accuracy for the entire 0 - 50% noise range, but even here the curve slopes upwards, nearing 0% at 50% noise. All of the curves, with the exception of the Thyroid curve, approach zero at 50% noise, thus at this noise level the algorithms have approximately equal classification accuracies.

When these results are taken together it can be seen that, for the Thyroid and Monks 3 problems, CN2-D produces less complex rulesets which have a

higher classification accuracy. For Monks 1 and Tic Tac Toe, the production by CN2 of more complex rulesets as noise is increased has a particularly dramatic effect on classification accuracy; so much that the simpler rulesets produced by CN2-D begin to match or outperform those of CN2. For Monks 2 and Hayes-Roth this effect is less marked; however the rulesets produced by CN2 on Monks 2 are always much more complex than those produced by CN2-D. These differences are much less for Hayes-Roth.

7.3.2 Further Analysis of Classification Accuracy

From the above description of the classification accuracy results it is clear that there are interactions found within the results for each learning problem.. In Figure 7.2 it can be seen that the curves describing the behaviour of CN2 and CN2-D for the Hayes Roth, Tic-Tac-Toe, Monks 2 and Monks 1 problems converge: i.e the performance of the two algorithms become similar at higher levels of noise. These results could be described in terms of floor effects: i.e. the algorithm reaches a floor which represents the poorest classification accuracy for that problem. Any further increase in noise will not bring about a corresponding decrease in classification accuracy. The algorithms appear to converge because CN2-D reaches this floor first.

However, this is not the case for Monks 3 and the Thyroid problem. Figure 7.2 shows that, for the Thyroid problem, CN2-D still has a higher classification accuracy than CN2 at 50% noise. The curve for Monks 3 has a complex shape in which the convergence shown by the four problems described above is from the positive region of the graph and also that the curve rises steeply from the noise free case for which both algorithms have similar

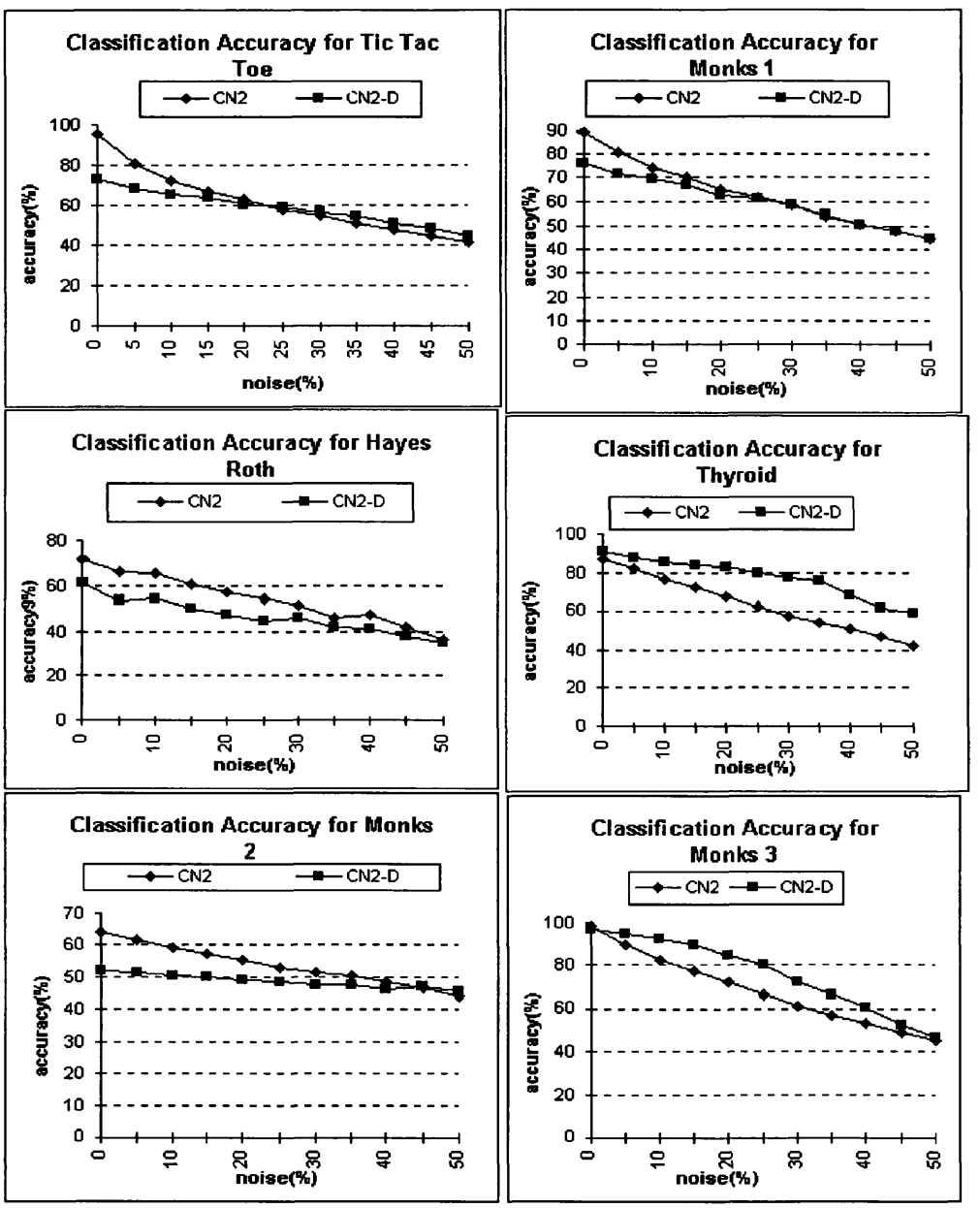

Figure 7.4: Classification accuracy

	noise level					
	0%		15%		30%	
Problem	t(df)	p	t(df)	p	t(df)	p
Monks 3	2.26(8)	0.054	-12.17(7)	0.0000	-5.75(4)	0.0045
Thyroid	-2.77(5)	0.039	-9.71(5)	0.0002	-11.65(4)	0.0003
Hayes Roth	2.58(7)	0.036	4.84(7)	0.002	4.02(7)	0.0052
Monks 1	6.86(6)	0.0002	3.87(7)	0.0063	0.07(8)	0.94
Tic-Tac-Toe	15.0(5)	0.0000	4.89(5)	0.0045	-1.5(8)	0.17
Monks 2	8.26(6)	0.0002	8.04(7)	0.0000	8.2(6)	0.0002

Table 7.2: t(df) and p scores for the six learning problems

(high) classification accuracy. To examine how the algorithms' behaviour changes with noise it was decided to perform two sample T-tests at 3 levels of noise: the noise free level, the 15% noise level and the 30% noise level. The p and t values from these analyses are given in table 7.2. The degrees of freedom used in each T-test are given in brackets after each t value.

The p scores given in table 7.2 show that there are significant differences in the performance of CN2 and CN2-D for most of the learning problems at the noise levels analysed. In this case a significant difference corresponds to a p-value which is less than 0.05. As discussed above, the general tendency for the classification accuracy to converge as noise increases because of the floor effect may explain the few results that were not significant at the 30% noise level. However the insignificant p-value obtained by Monks 3 for 0% noise is more likely to be caused by a ceiling effect: i.e. the two algorithms cannot

perform any better on this problem (both algorithms score close to 100%). In general using the t-tests confirms the observations discussed above: CN2-D performs better than CN2 on Monks 3 and Thyroid and more poorly than CN2-D on the rest. Hhowever if floor effects are considered, both algorithms perform poorly for the noisier (> 10%) levels of these problems.

If these algorithms were to be used to construct rules in real world problems where the amount of noise is unknown and the nature of the problem is less clear, a classification accuracy of less than 80% would be considered too poor to be useful. It would also be unusual to find a training set containing more than 15% noise. If the results are viewed within this context (see figure 7.2) then CN2 is shown to be less tolerant of noise than CN2-D. CN2 initially performs very well for the Monks 3, Monks 1, Tic-Tac-Toe and Thyroid, but by at most 10% noise its performance has dropped below 80%. CN2-D remains above 80% for both Thyroid and Monks 3 until the 25% noise level is reached. However CN2-D does not reach a classification of 80% for the other problems.

7.3.3 Coverage and Accuracy of Individual Rules

A characteristic of CN2-D is that the strategy of concentrating induction on easy examples produces a ruleset which does not correctly cover the training set. In this way incorrect information contained within the noisy examples in the training set can be avoided, and the ruleset will, paradoxically, perform with a higher classification accuracy on the test set. Results from selected noise levels and datasets concerning this characteristic are given in tables 7.3 - 7.6. These tables display the size, number and accuracy of rules which

coverage	no of rules		Ave No of conditions		Average training acc		Average test acc	
	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
0-5%	19	1	3.79	2	98.95	100	63.87	100
-10%	7	-	2.57	-	98.41	-	95.41	-
-15%	3	1	2	2	100	66.67	95.82	74.47
-20%	-	1	-	2	-	93.7	-	91.97
-25%	2	-	1.5	-	92.85	-	97.39	-
-40%*	-	2	-	1	-	86.4	-	88.73
total for ruleset	31	5	3.23	1.6	98.53	86.62	86.80	88.21

Table 7.3: Average classification accuracy and average number of conditions for rules of increasing % coverage induced from the Thyroid dataset with 5% noise (* there were no rules induced by any algorithm with coverage between 25% and 35%)

cover an increasing percentage of the training set. Each table corresponds to one single run of the experiment: tables 7.3 and 7.4 correspond to the 5 and 50% noise level for the Thyroid problem: tables 7.5 and 7.6 correspond to the same noise levels for the Monks 2 problem. These tables enable the ruleset statistics for a learning problem where CN2-D performed well to be compared with the statistics for a learning problem where CN2-D performed poorly.

Nearly 2/3 of the ruleset induced by CN2 from the Thyroid learning

coverage	no of rules		Ave No of conditions		Average training acc		Average test acc	
	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
0-5%	83	8	4.07	3.25	97.71	73.52	41.67	61.79
-10%	-	5	-	2.8	-	60.99	-	58.14
-15%	-	2	-	2.5	-	52.38	-	38.67
-20%	-	2	-	2	-	61.56	-	79.2
-25%	-	1	-	2	-	65.11	-	17.33
total for ruleset	83	18	4.07	2.83	97.71	65.90	41.67	47.65

Table 7.4: Average classification accuracy and average number of conditions for rules of increasing % coverage induced from the Thyroid dataset with 50% noise

problem at the 5% noise level is composed of rules which cover less than 5% of the training set. If $\leq$5% coverage is taken as the definition for a small disjunct, then a majority of this ruleset is composed of small disjuncts. In contrast, the CN2-D ruleset has only 5 rules: 6 times fewer than the CN2 ruleset. Only 1 of these rules is a small disjunct. The CN2 ruleset again achieves close to 100% accuracy on the training set, decreasing to 86.80% on testing. CN2-D achieves an accuracy of only 86.62% on the training set, but testing reveals an accuracy of 88.21%. This picture is repeated for the 50% noise level where the (large) CN2 ruleset is entirely composed of small disjuncts each with more than 4 conditions. 8 out of 18 rules for CN2-D are small disjuncts, and on testing CN2-D has c. 6% higher classification accuracy. So, for the Thyroid problem, inducing succinct rules which do not correctly cover the training set is a better strategy than the strategy of inducing more complex rulesets which correctly cover the training set. The low noise-free complexity for this problem guarantees that most of the difficult examples will be the misclassified examples added as noise, thereby enabling the induction process to avoid learning from them.

However, for Monks 2 the same strategy produces a poorer classification accuracy for CN2-D. At 5% noise, all 48 rules produced by CN2 covered less than 10% of the training set and had on average 3.47 conditions. In addition 41 of these rules were small disjuncts. In contrast CN2-D has only 4 small disjuncts out of 16 rules, and the average number of conditions per rule is 2.44. CN2 correctly covers the training set: and in this case achieves a test accuracy of 67.45%. As expected, CN2-D does not correctly cover the training set: it has a classification accuracy of only 68.75%. The test

coverage	no of rules		Ave No of conditions		Average training acc		Average test acc	
	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
0-5%	41	4	3.61	3	100	64.71	59.6	42.42
-10%	7	5	2.47	2.4	100	64.27	84.26	52.67
-15%	-	1	-	2.5	-	59.1	-	54.17
-20%	-	3	-	2	-	75	-	64.64
-25%	-	2	-	2	-	67.39	-	62.43
total for ruleset	48	16	3.47	2.44	100	68.75	67.45	57.06

Table 7.5: Average classification accuracy and average number of conditions for rules of increasing % coverage induced from the Monks-2 dataset with 5% noise

coverage	no of rules		Ave No of conditions		Average training acc		Average test acc	
	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
0-5%	50	3	3.46	2.67	99.26	69.23	49.75	48.48
-10%	-	7	-	2.43	-	66	-	51.21
-15%	-	2	-	2	-	70.83	-	51.96
-20%	-	5	-	2	-	68.09	-	44.60
total for ruleset	50	17	3.46	2.29	99.26	67.95	49.75	47.47

Table 7.6: Average classification accuracy and average number of conditions for rules of increasing % coverage induced from the Monks-2 dataset with 50% noise

classification accuracy falls from this to 57.06%. A similar picture emerges for the 50% noise level: in this case all of the rules induced by CN2 are small disjuncts: compared to 3 out 17 small disjuncts for CN2-D. CN2 has a near correct coverage of the training set, however this falls to 49.75% for the test set. This is only c 2% higher than CN2-D for this case. Thus for Monks 2 the induction of simple classifiers containing few small disjuncts is less successful than the strategy of inducing complex rulesets which correctly cover the training set. This is to be expected as the reduced training set for CN2-D will not contain sufficient information to allow induction of the correct rules. This is because the high noise-free complexity of this problem leads to a large number of correct but "awkward" examples to be omitted.

7.4 Discussion of Results

7.4.1 Intrinsic Difficulty and Classification Accuracy

The results described in the previous section reveal a complex situation in which the more succinct rulesets induced by CN2-D are better classifiers for some learning problems and poorer classifiers for other learning problems. It can be seen that the observed reduction in ruleset complexity does not bring about a corresponding and comparable decrease in classification accuracy.

Table 7.7 shows the noise free difficulty of the learning problems expressed as a percentage of the training set size and the average classification accuracy for the noise free case, the 10% noise level and the 20% noise level. This noise free difficulty is termed the *intrinsic* difficulty of the learning problem. The difficult examples found represent examples whose attribute-value pair

problem	intrinsic difficulty	noise level					
		0%		10%		20%	
		CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
Monks 3	7.37	98.36	97.08	82.81	92.93	72.23	84.34
Thyroid	14.5	86.97	90.68	76.73	85.7	67.71	82.86
HR	18.2	71.66	61.39	65.32	54.21	57.19	47.24
Monks 1	23.39	89.12	76.01	73.94	69.83	65.23	62.12
TTT	30.4	95.3	73.04	72.45	64.89	62.44	60.75
Monks 2	42.26	64.29	51.90	59.05	50.18	55.32	49.07

Table 7.7: Intrinsic difficulty and classification accuracies for selected noise levels of the learning problems studied

configurations are more representative of a classification other than that class to which they belong: thus this measure gives a good indication of the complexity of the problem before noise was added. As described in chapter 5, a rule-based algorithm such as CN2, using the generalisation/specialisation solution to the covering problem, will find it difficult to induce rules to cover these intrinsically difficult examples; giving very specialised rulesets with many long rules covering few examples. Even a statistically based stopping criterion designed to halt rule growth allowing a small number of negative examples to be covered cannot reduce ruleset complexity effectively. This is illustrated by the very much more complex rulesets induced by CN2 in all the learning problems studied and at all noise levels.

From table 7.7 it can be seen that CN2-D performs better than CN2

in learning problems which have a low intrinsic difficulty. (Thyroid and Monks-3). In these cases the strategy of avoiding difficult examples has allowed CN2-D to avoid most of the misclassified examples which become difficult when their classificationa are altered. The low intrinsic difficulty means that little crucial information is lost from difficult examples which are not misclassifications.

As expected, CN2-D has a poor classification accuracy score for Tic-Tac-Toe and Monks 2. These are learning problems that have representational complexity and a high intrinsic difficulty. In these cases the algorithms need to induce rulesets that correctly cover the difficult training examples so that the classes comprising the learning problem are fully described. The resulting complex rulesets are the only feasible way for attribute-value pair representations to do this. Adding noise to these circumstances will further complicate the induction process, with those training examples presented to CN2-D having inadequate information to describe the problem. As can be seen from table 7.7, CN2 has a higher classification accuracy than CN2-D for these problems.

CN2-D also has a poorer accuracy score than CN2 for Monks 1, a problem that has no representational complexity but has an intermediate intrinsic difficulty. In this case the moderate number of intrinsically difficult examples contained within the training set of Monks 1 contain enough information about the structure of the population to cause CN2-D to induce poorer rules. However CN2-D performs with the lowest classification accuracy on the Hayes-Roth problem which has only 4% more intrinsic difficulty than the Thyroid problem and has no representational complexity. This is due to the

unequal number of examples belonging to each class. This is described in the next section.

7.4.2 CN2-D, Noise and Unequally Proportioned Classes

Adding noise by altering the classification of training examples has a complex effect on the amount of difficult examples found in the new set. If the learning problem is a binary classification problem, (e.g. Tic-Tac-Toe and the Monks problems) then altering the classification of an example which was difficult before is likely to make that example not difficult. In this way noisy incorrect examples would be included in the reduced training set of CN2-D and induction would proceed from incorrect information. Changing the classification of an easy example is more likely to make it become difficult, thus induction would avoid this incorrect example. This situation is less likely with problems having more than two classes..

The result for the Hayes-Roth problem shows that CN2-D has a low classification accuracy on learning problems with multi-valued classification attributes where there is an unequal distribution of classes. It is a three-valued classification problem where the third class has substantially fewer members than the other two classes. In many cases the ruleset induced by CN2-D only contained rules for the two major classes: all of the training examples for the third class were identified as difficult, thus no rules were induced for this class. This accounts for the poor performance of CN2-D on this learning problem.

7.4.3 CN2-D and Representational Complexity

Both the Thyroid and Monks 3 problems are examples of the data characteristics outlined by Holte (1993). They are both very simple populations which, as the CN2-D results show, can be accurately described by a few general rules. When noise complicates this simple picture the strategy of ignoring difficult examples is sufficient to clear up some of the confusion that misclassification causes. However as more noise is added there is less correct information to learn from and thus the performance deteriorates. It is clear that CN2-D does not work well where there is initially a more complex problem: such as Monks 2 or Tic-Tac-Toe. In this case all of the information in the training set is needed, and identifying difficult examples cannot distinguish between noise and genuine complexity. However, if a relational learning system such as FOIL were used on these problems and the training examples were described using predicate logic, then these problems might also show the characteristics of simple datasets, within the context of the more descriptive knowledge representation. In this case a relational version of the d-score metric may allow FOIL to induce more succinct rules which perform more accurately on noisy data (see chapter 8). The Hayes Roth problem suggests that CN2-D does not perform well if there is a very unequal spread of class proportions within a problem. Thus CN2-D performs best on simple problems containing noisy examples.

The Thyroid problem, in which the more concise rules obtained by CN2-D had a higher classification accuracy than those obtained by CN2, for all noise levels, is the only problem to be considered that does not display con-

trived characteristics and is based on real data. The favourable results for this problem suggests that the strategy of only learning from easy training examples may work well in other real world problems.

The Thyroid and Monks 3 problems are also the problems where CN2-D remains above the 80% accuracy level until the 25% noise level. This is the level of accuracy required to produce a usable classifier, so in cases where noise has corrupted an otherwise straightforward learning problem the strategy of learning only from easy examples implemented by CN2-D produces more reliable and usable classifiers.

7.4.4 Summary

This chapter has discussed the results of experiments which were designed to test the hypothesis that identifying and avoiding difficult examples within noisy training sets produces less complex rulesets. In addition it was hypothesised that any reduction in ruleset complexity would not be matched by a corresponding decrease in classification accuracy. These experiments were conducted on a number of learning problems which displayed varying degrees and types of complexity. The results show that this strategy does produce less complex rulesets. In learning problems which have little intrinsic difficulty these rulesets are better classifiers than rulesets produced by CN2, which uses the statistical stopping criterion to overcome noise. Overall, induction of rules by CN2-D shows no decrease in classification accuracy which is comparable to the often substantial decrease in ruleset complexity. The concluding chapter to this thesis summarises the complexity reduction

strategy presented and the findings of the experiments. It then discusses some future work which could be used to extend the usefulness of the d-score metric.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

As described in Chapter 1, the aims of this thesis were to investigate the nature and behaviour of induction algorithms in differing learning situations. An investigation of the literature revealed that the induction of complex rulesets was a problem which diminished the utility of these algorithms. In particular, complex rulesets were induced from learning problems containing mistakes and/or representational complexity. This thesis presents a new approach to this problem, together with the results of experiments designed to evaluate the performance of the proposed approach.

Induction algorithms usually find some simple rules, but the success of this search can lead to the rest of the training set being described by very specific and complex rules. The new approach presented by this thesis is to reduce complexity by analysing the training set, so that misleading examples can either be identified and corrected, or the misleading information contained within them avoided by the induction algorithm. This was accom-

plished by a new metric called the d-score which analyses the attribute-value pairs of an example to identify if the example contains attribute-value pairs common to its class. If this is the case then that example is categorized as being 'easy' to learn from; if this is not the case then the example was categorised as being 'difficult' to learn from. To evaluate this approach, the d-score is used to preprocess training sets before they were passed to CN2. All 'difficult' examples were removed from the training sets and CN2 induced rules from only the easy examples. It was hoped that such a strategy might uncover the simple underlying structure of problem domains and CN2 would therefore induce simple but accurate rulesets. Preliminary experiments suggested that this new approach (termed CN2-D) always succeeded in inducing much less complex rulesets, and in noisy situations these rulesets were more accurate than those induced by CN2 from the full training set.

To further investigate the performance of CN2-D on noisy training sets, a series of more rigorous experiments have been carried out. A range of learning problems were used. These included problems with simple structure and problems which contained representational complexity. Noise was added in a controlled fashion so that the behaviour of both CN2-D and CN2 could be compared for a range of noise levels (0 - 50%). In all cases CN2-D produced much simpler rulesets than CN2. Indeed in some cases the rulesets had up to 90% fewer conditions per class. In two of the six problems these simpler rules also had a higher classification accuracy for all but one of the noise levels investigated. In another two problems the simpler rulesets had an inferior classification accuracy to the rulesets induced by CN2. For the remaining two problems CN2 had a higher classification accuracy for lower levels of

noise but this accuracy decreased at such a rate that CN2-D had a higher classification accuracy for noise levels greater than c 25%.

The results indicate that the approach of focusing inductive learning on easy, representative examples produces rulesets which are substantially less complex than those induced from the full training set. In noisy circumstances where there is no additional difficulty in the form of representational complexity, the simpler rulesets had comparable or superior classification accuracy. In cases where representational complexity was an additional complication, the simpler rules had an inferior classification accuracy. However, in no case was a substantial reduction in rule complexity matched by a comparable reduction in classification accuracy. However the poor performance of CN2-D on one problem which also did not contain representational complexity may point to the existence of other learning factors such as the distribution of classes in the problem: this specific learning problem (Hayes-Roth) has a very small number of examples belonging to one particular class.

The d-score metric presented in this thesis is therefore a reliable method for identifying unrepresentative examples in a training set. This can be seen by the substantial decrease in ruleset complexity observed when these examples are taken out of the training set. Any deterioration in classification accuracy of the simpler rulesets is never comparable to the reduction in ruleset complexity. Indeed, if the unrepresentative examples are wrongly classified examples, the simpler rulesets often have a higher classification accuracy. However, if the unrepresentative examples are truly unusual but correct examples of a class, the simple rulesets induced by their removal will perform with poorer classification accuracy.

8.2 Future Work

A more comprehensive set of experiments is required to fully evaluate the effect of applying CN2-D, with particular reference to how it behaves when processing problems containing an unequal distribution of classes. The Hayes-Roth result described above suggests that CN2-D may not perform as well in these circumstances. Further work may explain this phenomenon.

Also, the d-score metric is at present limited to training sets containing nominal attributes which are entirely described using propositional attributes. As described in chapter 6, the d-score metric works out the relative frequency for all the attribute-value pairs for each class in the problem, combines these values for the attribute-value pairs comprising an example and yields a score for each example, which allows unrepresentative examples to be identified.

This process is problematical for a numerical attribute but one possible solution is to partition the range of values found into discrete subsets. Each subset can then be treated as a single category in the same way as single values of a nominal attribute. However it may be difficult to decide on how many subsets to have, and therefore how wide the range would be. This partitioning should be independent of class distribution, and so the binary partitioning of numerical attributes performed by CART (Brieman et al 1984) would be inappropriate. An unbiased partitioning may be achieved by obtaining a frequency distribution of the values of the numerical attribute and choosing ranges each of which which had an equal frequency. The ranges of such a partition would be unequal, but the class distribution would not

be changed by this partitioning and so numerical attributes may be treated similarly to nominal attributes.

In chapter 5, it was mentioned that FOIL (Quinlan 1990) also suffers from the small disjuncts problem because it also uses the "separate and conquer" method of growing rules. At the moment, the d-score metric works only with learning problems entirely described by attribute value pairs. To investigate whether simpler rules might be induced by FOIL from the pre-processed training sets produced by the d-score metric, a predicate version of the d-score metric could be developed. This would work in a rather different way to the analysis of attribute value pairs. Relative frequency could be evaluated for each constant in the extensional definition of the target predicate and also for extensional definitions of the background predicates. Examples could be scored by combining the relative frequencies for each constant found in the example, for each class (only true and false in this case). Difficult examples could be identified in a similar way to that used for attribute value pairs. An example in the POS set with a higher score for the value "false" would be one case where an difficult example would be found.

These two improvements would allow this technique to be more thoroughly examined in experiments where the whole population cannot be generated and so training sets and test sets would have to be sampled from the population. This would also allow the method to be compared with a greater number of machine learning studies. The extension of the method to predicate logic may also overcome limitations which are a result of the inadequacy of using an attribute value representation for learning problems in which the relationships between problem components describe the problem

more efficiently (e.g. the Tic-Tac-Toe problem).

8.2.1 Concluding Remarks

The research work central to this thesis has illustrated how concentrating on simple patterns can give a result which is easier to understand and as reliable as results obtained from complex patterns.

> *Getsuin said to his students "Keichu, the first wheelmaker of China, made two wheels of fifty spokes each. Now suppose you removed the nave uniting the spokes ? "*

References

Aha D.W (1991) Incremental Constructive Induction: An Instance-based Approach. Proceedings of the Eighth International Workshop on Machine Learning Evanston Ill Morgan Kaufmann pp 117 - 121.

Aha D.W (1992) Tolerating Noisy, Irrelevant and Novel Attributes in Instance Based Learning Algorithms. Int J of Man Machine Studies 36 pp 267 - 287.

Aha D.W, Kibler D, Albert M.K. (1991) Instance Based Learning Algorithms. Machine Learning 6 pp 37 - 66.

Ali K.M, Pazzani M.J (1993) HYDRA: A Noise-Tolerant Relational Concept Learning Algorithm. IJCAI 13.

Ali K.M, Pazzani M.J (1993) Reducing the Small Disjuncts Problem by Learning Probablistic Concept Descriptions. Computational Learning Theory and Natural Learning Systems 3 Cambridge, Mass MIT Press.

Barnden J, Srinivas K. (1992) Overcoming Rule Based Rigidity and Connectionist Limitations Through Massively Parallel Case Based Reasoning. Int J of Man Machine Studies 36 pp 221 - 246.

Bergedano F, Matwin S, Michalski R.S, Zhang J. (1992) Learning Two Tiered Descriptions of Flexible Concepts - The POSEIDON system. Machine Learning 8 pp 5 - 43.

Brieman L, Friedman J.H, Olshen R.A, Stone C.J (1984) Classification and Regression Trees. New York Chapman & Hall.

Buchanan B, Feigenbaum E (1978) DENDRAL and Meta-DENDRAL: Their Applications. Dimension Artificial Intelligence 11.

Bunk C, Pazzani M.J (1991) An Investigation of Noise-Tolerant Relational Learning Algorithms. Proceedings of the Eighth International Workshop on Machine Learning Evanston Morgan Kaufmann.

Callan J.P, Utgoff P.E (1991) Constructive Induction on Domain Information. Proceedings of the Ninth National Conference on Artificial Intelligence. Anaheim CA MIT Press pp 614 - 619.

Cendrowska (1987) PRISM: An Algorithm for Inducing Modular Rules. Int J of Man-Machine Studies 27, 349-370.

Cestnik B, Kononenko I, Bratko I (1987) Assistant 86: A Knowledge-Elicitation Tool for Sophisticated Users. Progress in Machine Learning Bled Sigma Press.

Chi R.T.H Kiang M.Y. (1992) Knowledge Acquisition from an Incomplete Domain Theory - An Application on the Stock Market. Computer Sc in Econ and Man 8 pp 1 - 21.

Clark P, Niblett T. (1988) The CN2 Induction Algorithm. Machine Learning 3 pp 261 - 283.

Cohen W.W, (1992) Efficient Pruning Methods for Separate-and-Conquer Rule Learning Systems. AT&T Bell Labs Technical Memorandum.

De Raedt L, Bruynooghe M (1992) Belief Updating from Integrity Constraints and Queries. Artificial Intelligence 53, pp 291 - 307.

De Raedt L, Bruynooghe M (1994) Interactive Theory Revision. Machine Learning: A Multistrategy Approach San Francisco Morgan Kaufmann.

Duda R.O, Hart P, Konolige K Reboh R (1979) A Computer-Based Consultation for Mineral Exploration. Stanford Research Institute Report.

Durkin J (1994) Expert Systems: Design and Development. New Jersey Prentice Hall.

Esposito F, Malerba D, Semeraro G (1992) Classification in Noisy Environments Using a Distance Measure Between Structural Symbolic Descriptions. IEEE Transactions on Pattern Matching and Machine Learning 14 (3) pp 390 - 402.

Fayyad U.M, Irani K.B (1990) What Should Be Minimised in a Decision Tree ? Proceedings of the Eighth National Conference on Artificial Intelligence AAAI-90.

File P. E, Dugard P. I, Houston A. S (1994) Evaluation of the Use of Induction in the Development of a Medical Expert System Computers and Biomedical Research 27, 383 - 395

Gemello R, Mana F, Saitta L. (1991) RIGEL : An Inductive Learning System. Machine Learning 6 pp 7 - 35.

Green C.C (1969) The Application of Theorem Proving to Question Answering Systems. IJCAI 1 pp 219 - 237.

Hayes-Roth F, Waterman D.A, Lenat D.B (1983) Building Expert Systems. Reading Addison Wesley.

Hirsh H (1992) Polynomial-Time Learning with Version Spaces. Proceedings of AAAI 92 San Jose pp 117 - 122 Systems. Reading Addison Wesley.

Holte R.C. (1993) Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. Machine Learning 11 pp 63 - 91.

Holte R.C, Acker L.E, Porter B.W. (1989) Concept Learning and the Problem of Small Disjuncts. IJAI 89 pp 813 - 818.

Hong J. AE1: (1985) An Extension Matrix Approximate Method for the General Covering Problem. International Journal of Computer and Information Sciences 14 (6) pp 421 - 437.

Kolodner J.L. (1992) An Introduction to Case Based Reasoning. Artificial Intelligence review 6 pp 3 - 34.

Michalski R.S. (1980) Pattern Recognition as Rule-Guided Inductive Inference. IEEE Transactions on Pattern Matching and Machine Intelligence 2 (4) pp 349 - 361.

Michalski R.S. (1983) A Theory and Methodoloy of Inductive Learning. Machine Learning - an A.I. approach 1 pp 83 - 129.

Michalski R.S. (1994) Inferential Theory of Learning: Developing Foundations for Multistrategy Learning. MachiNe Learning: A Multistrategy Approach San Francisco Morgan Kaufmann.

Michalski R.S. Chilauski R.L. (1980) Knowledge Acquisition by Encoding Expert Rules versus Computer Induction from Examples: a case study involving soybean pathology. Jrnl of Man-Machine studies 12. pp 63 - 87

Michalski R.S. Larson J.B. (1978) Selection of the Most Representative Training Examples and Incremental Generation of VL1 Hypothesis: the Underlying Methodology of Programs Esel and AQ11. Technical Report 877 Dept of Computer Science U. Illinois Urbana Champaign.

Michalski R.S, Mozetic I, Hong J, Lavrac N. (1986) The Multi-Purpose Incremental Learning System AQ15 and its Application to Three Medical Domains. Proceedings of the 5th AAAI pp 1041-1045.

Michie D, Spiegelhalter D.J, Taylor C.C. (1994) Machine Learning, Neural and Statistical Classification. London Ellis Horwood.

Milne R, Nicol C, Trave-Massuyes L, Quevedo J (1995) TIGER: Knowledge Based Gas Turbine Condition Monitoring. Proceedings of Expert Systems 95 Oxford SGES.

Mitchell T.M. (1982) Generalisation as Search. Artificial intelligence vol 18(2) pp 203 - 226

Mooney R.J, Califf M.E (1995) Induction of First-Order Decision Lists: Results on Learning the Past Tense of English Verbs. Journal of Artificial Intelligence Research 3 pp 1 - 24.

Mooney R, Ourston D (1989) IOU - Integrated Learning of Concepts with both Explainable and Conventional aspects. Proc of 6th Int Work on Machine Learning 5-7.

Muggleton S, Feng C (1990) Efficient Induction of Logic Programs. Proceedings of the First Conference on Algorithmic Theory Tokyo.

Murphy P, Aha D (1994) UCI Repository of Machine Learning Databases. Irvine, CA: University of California, Department of Information and Computer Science.

Murphy P.M, Pazzani M.J (1994) Exploring the Decision Forest: An Empirical Investigation Of Occam's razor in Decision Tree Induction. Journal of Artificial Intelligence Research 1 pp 257 - 275.

Niblett T, Bratko I. Learning Decision Rules in Noisy Domains. Turing Institute Technical Report.

Newall A, Shaw J.C, Simon H.A (1960) A Variety of Intelligent Learning in a General Problem Solver. Self Organising Systems New York Pergamon Press.

Pazzani M.J, Kibler (1992) The Utility of Knowledge in Inductive Learning. Machine Learning 9 pp 57 - 94.

Quinlan J.R (1986) Induction of Decision Trees. Machine Learning 1 pp 81 - 106.

Quinlan J.R (1990) Learning Logical Definitions from Relations. Machine Learning 5 pp 239 - 266.

Quinlan J.R (1993) C4.5: Programs for Machine Learning. San Mateo Morgan Kaufmann.

Rymon R (1993) An SE-Tree based Characterisation of the Induction Problem. Proceedings of the Tenth International Conference on Machine Learning Amherst MA pp 268 - 275.

Rymon R (1996) SE-trees Outperform Decision Trees in Noisy Domains. Proceedings of the Second Conference on Knowledge Discovery in Databases Portland OR.

Shannon C.E (1955) A Chess Playing Machine. The World of Mathematics 4 New York.

Shortcliffe E.H (1976) Computer-Based Medical Consultation: MYCIN. New York American Elsevier.

Slade S. (1991) Case Based Reasoning - A Research Paradigm. Artificial Intelligence Magazine Spring pp 42 - 55.

Thrun S.B, Bala J, Bloedorn E, Bratko I, Cestnik B, Cheng J, De Jong K, Dzeroski S, Fahlman S.E, Fisher D, Hamann R, Kaufman K, Keller S, Kononenko I, Kreuziger J, Michalski R.S, Mitchell T, Pachowicz P, Reich Y, Vafaie H, Van de Welde W, Wenzel W, Wnek J, Zhang J (1991) The Monks Problems — A Performance Comparison of Different Learning Algorithms. Carnegie Mellon University report CMU-CS-91-197.

Webb G.I, (1996) Further Experimental Evidence Against the Utility of Occam's Razor. Journal of Artificial Intelligence Research 4 pp 397 - 417.

Wnek J, Michalski R.S. (1994) Hypothesis-Driven Constructive Induction in AQ17-HCI: A Method and Experiments. Machine Learning 14 pp 139 - 168.

Wnek J, Michalski R.S (1994) Comparing Symbolic and Subsymbolic Learning. Machine Learning: A Multistrategy Approach San Francisco Morgan Kaufmann.

Wu X (1993) Inductive Learning: Algorithms and Frontiers. Artificial Intelligence Review, 7 (2) pp 93 - 108.

Wusteman J. (1992) Explanation Based Learning - A Summary. Artificial Intelligence Review 6 pp 243 - 262.

Zeleznikow J, Stranieri A, Lewis B (1995) Using Induction in Legal Expert Systems. Proceedings of Expert Systems 95 pp 101 - 114.

Appendix 1: Tabulated Results

	1		2		3		4		5		Average	
noise	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
0	97.87	76.67	94.15	70.7	95.12	75.48	94.9	72.54	94.5	69.8	95.308	73.038
5	80.943	67.425	81.963	66.263	79.36	69.218	81.137	69.337	80.281	67.509	80.737	67.95
10	72.482	65.038	74.34	65.124	71.831	64.624	72.337	64.727	71.289	64.943	72.456	64.891
15	66.389	64.341	67.205	64.413	67.176	63.316	67.365	64.311	66.169	61.016	66.861	63.479
20	62.495	61.912	63.422	59.779	63.472	61.888	61.378	59.583	61.41	60.595	62.435	60.751
25	58.008	59.065	59.042	58.093	58.633	59.746	57.266	58.779	58.158	58.655	58.221	58.868
30	53.708	55.452	56.12	58.34	55.248	58.114	54.765	55.406	55.526	54.582	55.073	56.379
35	51.688	56.503	51.15	51.651	50.384	54.586	51.119	55.262	50.873	55.422	51.043	54.685
40	48.655	51.472	47.215	51.434	47.706	50.386	49.303	50.89	47.104	51.512	47.997	51.139
45	45.29	49.105	44.751	49.801	45.034	48.977	43.935	47.703	45.566	49.186	44.915	48.954
50	42.644	46.817	41.522	46.072	41.923	43.365	42.203	43.521	42.337	43.676	42.126	44.69

Table 8.1: Classification accuracy for Tic Tac Toe

	1		2		3		4		5		Average	
noise	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
0	32	26	47.5	28.5	34.5	27	39	30	38	25.5	38.2	27.4
5	76.1	28.1	82.85	32.2	80.75	30	77.45	28.95	78.7	28.75	79.17	29.6
10	103.2	29.4	103.15	31.7	103.75	32.65	99.5	32.65	102.3	32.85	102.38	31.85
15	118.55	32.45	121.55	32.6	118.75	35.4	122.85	34.15	121.7	31.85	120.68	33.29
20	129.05	33.8	129.55	35.4	128.5	31.35	127.9	34.8	127.05	34.95	128.41	34.06
25	135.7	32.05	144.45	33.6	138.1	36.6	137.25	33.8	141.55	35.2	139.41	34.25
30	143.25	36	142.35	33.9	143.35	34.85	144.6	36.2	142.7	34.55	143.25	35.1
35	148	32.85	150.3	38.15	147.8	34.25	153.1	36.65	150.4	31.9	149.92	34.76
40	146.35	38.25	149.15	36.15	149.5	37.15	153.3	35	154.3	33.8	150.52	36.07
45	149.05	33.35	151.15	33.55	154.45	36.7	150.3	35.65	151.75	36.4	151.34	35.13
50	151.85	34.4	153.95	35.8	150.3	36	159.85	35.65	155.25	36.5	154.24	35.67

Table 8.2: Ruleset Complexity for Tic Tac Toe

	1		2		3		4		5		Average	
noise	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
0	77.72	68.18	62.71	57.7	68.56	61.74	77.39	52.9	71.9	66.41	71.656	61.386
5	68.932	62.985	57.66	50.473	66.029	53.465	70.804	49.981	67.891	52.147	66.263	53.81
10	66.584	56.092	57.122	54.439	67.109	52.368	70.843	50.326	64.944	57.809	65.32	54.207
15	60.576	56.055	56.478	50.335	63.427	47.973	62.671	49.499	58.013	46.623	60.233	50.097
20	60.969	48.252	49.365	47.404	56.862	50.853	59.919	45.525	58.834	44.186	57.19	47.244
25	55.588	49.958	46.963	40.498	55.27	47.355	57.226	42.455	54.614	41.964	53.932	44.446
30	52.634	46.473	50.07	45.587	47.886	43.542	53.987	48.48	51.409	45.255	51.197	45.867
35	48.065	42.228	44.424	41.534	44.528	41.873	47.245	44.692	46.99	38.492	46.25	41.764
40	55.778	47.745	42.905	38.537	47.194	43.14	45.892	42.912	44.084	34.012	47.171	41.269
45	44.404	37.972	34.56	34.439	41.093	34.526	46.253	42.96	40.879	36.606	41.438	37.301
50	38.486	36.936	34.614	33.805	36.152	39.437	34.774	33.086	35.716	32.256	35.948	35.104

Table 8.3: Classification accuracy for Hayes Roth

	1		2		3		4		5		Average	
noise	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
0	7	5	7.66	5.33	9	7.33	9.66	5	10	5	8.664	5.532
5	8.598	3.398	8.295	5.23	10.031	5.93	11.896	4.996	11.231	5.031	10.01	4.917
10	8.366	4.13	11.129	5.33	10.862	4.43	11.528	3.498	12.232	5.962	10.823	4.67
15	9.095	4.696	12.266	5.161	10.364	4.498	12.964	3.631	10.398	3.862	11.017	4.3696
20	11.63	3.796	12.199	4.229	14.497	4.496	13.763	3.732	11.231	4.932	12.664	4.237
25	9.164	4.697	13.666	5.929	13.394	5.465	14.497	3.764	12.862	3.897	12.717	4.7504
30	11.732	4.664	12.932	4.031	11.795	4.228	13.494	2.996	14.962	4.43	12.983	4.0698
35	11.863	4.031	13.262	3.496	14.132	5.431	14.332	3.265	12.497	3.464	13.217	3.9374
40	11.929	3.83	14.294	3.96	12.398	3.629	13.496	2.196	13.563	2.755	13.136	3.274
45	13.264	3.429	12.029	4.366	12.529	3.595	13.262	3.162	13.764	3.029	12.97	3.5162
50	13.796	4.597	13.096	4.596	14.532	3.898	12.462	2.998	13.93	2.827	13.563	3.7832

Table 8.4: Ruleset Complexity for Hayes Roth

	1		2		3		4		5		Average	
noise	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
0	63.98	48.45	63.13	55.67	67.36	53.83	62.97	50.73	64.02	50.84	64.292	51.904
5	61.026	50.364	60.513	50.08	61.492	52.663	64.046	52.044	61.471	51.169	61.71	51.264
10	58.681	51.11	58.075	49.416	58.804	49.632	61.263	51.906	58.432	48.82	59.051	50.177
15	57.612	48.844	57.243	50.381	57.942	51.007	59.594	51.428	54.995	48.781	57.477	50.088
20	54.211	49.598	54.81	48.541	57.572	49.02	56.009	50.339	54.003	47.863	55.321	49.072
25	53.617	47.98	53.111	48.114	52.358	49.444	53.497	48.763	52.019	49.172	52.92	48.695
30	51.979	47.569	51.37	47.306	51.32	47.881	51.788	46.558	49.607	47.321	51.213	47.327
35	51.144	47.701	50.226	45.621	49.893	49.525	51.737	47.487	48.87	48.297	50.374	47.726
40	48.499	46.75	47.417	45.854	49.065	47.657	48.001	45.676	49.187	45.152	48.434	46.218
45	47.131	48.74	46.184	47.249	46.716	46.504	45.28	45.067	46.815	46.599	46.425	46.832
50	44.828	45.761	43.563	45.336	45.082	47.381	43.099	44.052	43.643	44.368	44.043	45.38

Table 8.5: Classification accuracy for Monks 2

	1		2		3		4		5		Average	
noise	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
0	82.5	13	90.5	19	80	18.5	72.5	17.5	69.5	22	79	18
5	87.2	18.6	85.95	17.55	81.4	16.5	77.35	16.15	75	20.65	81.38	17.89
10	82.2	14.6	87.4	18.4	85.4	17	77.25	15.9	76.5	17.65	81.75	16.71
15	86.1	18.4	86.65	20.25	83.2	19.9	81.25	18.6	80.7	17.9	83.58	19.01
20	81.15	16.2	82.65	18.75	88.85	19.1	83.75	20.3	80.45	16.9	83.37	18.25
25	85.35	17.6	84.5	18.8	86.35	17.95	85.05	16.65	85.95	18.45	85.44	17.89
30	86.2	19.6	87.25	18.1	90.2	20.3	82.5	19.3	88.25	16.75	86.88	18.81
35	87.05	15.9	89.55	19.8	94.2	20.35	88.5	16.9	83.8	17.95	88.62	18.18
40	86.45	18.35	85.35	17.85	86.35	20.75	87.05	20.45	87.05	20.1	86.45	19.5
45	79.5	18.45	89.4	21.35	87.2	20.85	87.55	19.1	88.9	18.4	86.51	19.63
50	90.5	18.5	87.25	17.2	87.25	18.3	82.8	20.75	87.1	18.9	86.98	18.73

Table 8.6: Ruleset complexity for Monks 2

	1		2		3		4		5		Average	
noise	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
0	86.07	75.44	90	75.46	86.34	77	90.74	71.64	92.47	80.55	89.124	76.018
5	81.746	73.317	82.817	71.118	78.36	70.599	82.3	73.511	79.172	71.022	80.879	71.913
10	75.147	71.73	73.541	68.728	71.089	70.018	75.576	70.188	74.363	68.507	73.943	69.834
15	72.545	67.673	69.993	64.682	68.877	67.178	70.511	67.859	69.922	67.889	70.37	67.056
20	67.001	62.157	64.701	61.114	65.211	63.996	64.473	63.01	64.791	60.304	65.235	62.116
25	62.283	63.134	60.106	59.062	61.586	61.64	63.495	64.075	60.212	57.134	61.536	61.009
30	59.583	55.688	56.751	56.546	58.63	59.245	57.285	62.079	60.454	58.664	58.541	58.444
35	53.772	54.685	52.517	53.452	54.013	52.833	54.193	58.685	53.681	53.987	53.635	54.728
40	51.399	48.814	50.513	51.697	49.195	51.52	51.909	51.303	50.179	51.059	50.639	50.879
45	46.865	49.454	47.88	45.701	49.398	50.78	48.039	47.817	49.19	47.581	48.274	48.267
50	44.217	45.75	44.529	43.765	45.827	45.857	44.252	45.805	43.261	42.794	44.417	44.794

Table 8.7: Classification accuracy for Monks 1

	1		2		3		4		5		Average	
noise	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
0	29	14.5	31	8	24	10	25	10.5	30.5	6.5	27.9	9.9
5	40.25	15.8	44.65	10.5	36.75	15.35	37.6	12.95	43.95	7.65	40.64	12.45
10	53.35	15.5	55.3	14.75	53.7	16.8	50.35	13.65	53.2	12.25	53.18	14.59
15	63.6	18.85	64.55	17.75	67	16.75	60.4	14.15	59.75	13.5	63.06	16.2
20	67.25	17.65	71.95	19.5	72.1	16.3	70.8	19.45	69.55	17.45	70.33	18.07
25	73.65	18.65	78.85	18.55	76.9	20.65	73.6	18.85	74.8	18.85	75.56	19.11
30	85.15	18.85	87.05	19	75.55	19.2	78.6	20.4	79.65	18.75	81.2	19.24
35	84.5	20.6	78.15	22.45	78.8	16.1	81.65	20.5	83.55	20.1	81.33	19.95
40	88.55	20.95	86.45	19.05	83.4	19.55	82.15	18.55	87.25	20.9	85.56	19.8
45	82.4	20.95	82.85	18.1	88.2	19.4	89.65	19.05	83.35	19.35	85.29	19.37
50	86.9	19.45	92.75	19.05	86.55	17.45	90.55	20.85	85.75	19.75	88.5	19.31

Table 8.8: Ruleset complexity for Monks 1

	1		2		3		4		5		Average	
noise	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
0	85.04	89.16	84.21	90.17	86.36	91.67	87.93	91.58	91.31	90.81	86.97	90.678
5	84.371	88.871	83.662	86.778	81.648	87.492	81.504	87.492	81.141	87.555	82.465	87.638
10	77.22	87.695	78.255	84.73	75.861	84.289	77.132	84.289	75.188	87.496	76.731	85.7
15	72.867	85.362	72.625	80.134	72.311	82.837	72.809	82.837	70.466	86.175	72.216	83.469
20	69.158	85.077	68.666	77.233	67.89	83.326	66.822	83.326	66.026	85.342	67.712	82.861
25	62.022	78.472	65.276	78.364	61.757	79.899	62.767	79.899	62.744	83.114	62.913	79.95
30	58.539	76.261	58.176	71.91	57.972	79.54	58.951	79.54	55.932	80.751	57.914	77.6
35	56.058	76.645	54.751	70.407	54.065	77.346	55.316	77.346	53.627	75.815	54.763	75.512
40	51.029	64.374	52.063	65.459	50.874	69.416	50.766	69.416	50.345	72.199	51.015	68.173
45	46.9	61.098	47.456	55.887	47.347	63.156	47.359	63.156	45.531	65.316	46.919	61.723
50	44	50.9	43.332	53.491	40.94	63.059	41.853	63.059	41.936	64.289	42.412	58.96

Table 8.9: Classification accuracy for Thyroid

	1		2		3		4		5		Average	
noise	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
0	29.32	3.66	25.33	2.33	33.66	4.67	26.66	6	17.33	2.33	26.46	3.798
5	41.566	6.234	40.53	7.263	42.86	6.563	39.192	6.461	37.258	3.731	40.281	6.0504
10	57.733	6.501	51.795	8.23	57.625	8.527	51.163	6.163	53.993	6.028	54.462	7.0898
15	67.633	8.694	65.398	9.529	70.426	8.856	66.229	6.865	68.563	6.563	67.65	8.1014
20	81.534	7.455	82.397	9.964	80.063	8.454	79.592	8.43	80.46	7.03	80.809	8.2666
25	85.833	9.791	87.697	9.264	92.33	9.855	88.932	11.163	88.13	10.763	88.584	10.167
30	99.032	9.691	97.031	11.73	103	11.349	96.465	10.43	99.395	9.063	98.984	10.453
35	99.834	11.21	101.73	12.529	102.33	12.733	102.53	12.129	102.76	12.864	101.84	12.293
40	104.73	14.26	106.13	13.496	106.43	14.746	105.16	11.464	106.93	13.53	105.88	13.499
45	107.43	14.543	109.56	14.061	110.93	13.966	104.66	12.029	108.97	13.762	108.31	13.672
50	108.23	14.397	107.27	12.495	110.16	13.919	108	11.997	106.13	15.162	107.96	13.594

Table 8.10: Ruleset complexity for Thyroid

	1		2		3		4		5		Average	
noise	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
0	99.3	97.22	97.91	97.22	100	97.22	97.45	96.52	97.15	97.22	98.362	97.08
5	90.693	95.415	88.427	95.502	89.065	90.981	91.018	96.822	89.076	94.764	89.656	94.697
10	82.228	93.815	82.614	92.318	83.789	91.29	83.513	95.941	81.917	91.272	82.812	92.927
15	76.81	88.965	75.876	89.353	76.114	88.807	79.255	92.442	77.328	87.519	77.077	89.417
20	71.415	85.102	72.636	85.439	73.469	83.089	72.696	86.411	70.966	81.656	72.236	84.339
25	65.992	81.758	67.465	79.354	63.888	78.098	67.043	80.216	68.481	80.276	66.574	79.94
30	61.141	76.809	62.079	70.648	61.055	72.96	61.108	74.345	61.589	65.872	61.394	72.127
35	55.219	71.176	56.507	65.491	58.006	67.3	56.793	64.859	57.524	65.025	56.81	66.77
40	52.331	59.387	53.131	61.392	53.086	59.699	53.433	61.527	54.089	60.729	53.214	60.547
45	48.88	53.291	48.257	50.148	49.563	52.422	50.134	54.644	48.561	53.126	49.079	52.726
50	43.15	43.867	45.583	45.059	45.181	48.025	45.716	50.136	45.567	46.089	45.039	46.635

Table 8.11: Classification accuracy for Monks 3

	1		2		3		4		5		Average	
noise	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D	CN2	CN2-D
0	12.5	7	11.5	7	11.5	7	12	10.5	11.5	7	11.8	7.7
5	31.5	9.2	33.1	8.9	29.55	10.95	29.55	10.95	29	10.6	30.54	10.12
10	45.15	10.45	47.85	12.75	49.4	9.7	49.4	9.7	45.6	12.75	47.48	11.07
15	55.15	13.1	60.95	16.9	58.75	12.25	58.75	12.25	59.5	14.85	58.62	13.87
20	69.4	17.1	70.2	19	72.55	16.15	72.55	16.15	68.8	17.15	70.7	17.11
25	79.3	19.15	82.15	21.2	79.15	15.6	79.15	15.6	83.3	18.9	80.61	18.09
30	89.3	21.6	92.85	24.1	84.45	19.75	84.45	19.75	86.75	22.4	87.56	21.52
35	96.65	20.85	95.95	22.3	92.65	17.05	92.65	17.05	97.5	20.7	95.08	19.59
40	95.35	21.3	101.6	21.35	99.65	19.1	99.65	19.1	98.4	21.35	98.93	20.44
45	103.25	19.25	99.75	18.9	102.85	19.35	102.85	19.35	100.8	20.5	101.9	19.47
50	104.85	21.8	102.05	20.1	102.65	24.3	102.65	24.3	103.8	24.05	103.2	22.91

Table 8.12: Ruleset complexity for Monks 3